VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROCEDURÁLNE GENEROVANIE FAKTOROV STRACHU

PROCEDURAL GENERATION OF FEAR FACTORS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TERÉZIA HUNDÁKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MILET, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146196

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Studentka:	**Hundáková Terézia**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Procedurální generování faktorů strachu**
Kategorie:	Počítačová grafika
Akademický rok:	2022/23

Zadání:

1. Nastudujte tvorbu počítačových her a herní enginy. Seznamte se s teorií o lidském strachu a tvorbou hororových her.
2. Navrhněte aplikaci, která umožní procedurální generování herních úrovní, ve kterých se objevují herní mechaniky určené k vyvolání strachu.
3. Implementujte navrženou aplikace.
4. Vytvořenou aplikaci proměřte na uživatelích a vyhodnoťte.
5. Vytvořte demonstrační video a aplikaci zveřejněte.

Literatura:

- Gregory, Jason. *Game engine architecture*. crc Press, 2018. ISBN 1351974289, 9781351974288
- Bishop, Lars, et al. "Designing a PC game engine." IEEE Computer Graphics and Applications 18.1 (1998): 46-53.
- Adams, Ernest, and Joris Dormans. Game mechanics: advanced game design. New Riders, 2012. ISBN 0321820274, 9780321820273

Při obhajobě semestrální části projektu je požadováno:
Body jedna a dva a kostra aplikace.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Milet Tomáš, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	31.7.2023
Datum schválení:	31.10.2022

Abstrakt

Cieľom tejto práce je vytvoriť herné hororové demo v Unity s procedurálne generovaným prostredím a udalosťami. Práca skúma trh aktuálnych riešení, zaoberá sa teóriou strachu a herného vývoja. Ďalej popisuje do detailu navrhnuté herné mechaniky a spôsob, akým boli tieto mechaniky implementované. Výsledkom práce je herné demo s procedurálne generovaným labyrintom, v ktorom sa hráč snaží z tohto labyrintu utiecť. Po ceste naňho číhajú rôzne druhy strašidelných nástrah.

Abstract

The aim of this thesis is to create a horror game demo in Unity with procedurally generated environment and events. The work examines the market of current solutions and deals with the theory of fear and game development. It also describes designed mechanics of the game in detail and the way they have been implemented. The result of the work is a game demo with procedurally generated labyrinth, from where the player tries to escape. All kinds of scary traps await for them along the way.

Klíčová slova

hra, herné demo, Unity, procedurálne generovanie, labyrint, horor, faktory strachu

Keywords

game, game demo, Unity, procedural generation, maze, horror, fear factors

Citace

HUNDÁKOVÁ, Terézia. *Procedurálne generovanie faktorov strachu.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet, Ph.D.

Procedurálne generovanie faktorov strachu

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Ing. Tomáša Mileta, Ph.D. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Terézia Hundáková
29. července 2023

Poděkování

Chcela by som poďakovať vedúcemu práce doktorovi Tomášovi Miletovi za vedenie práce, rady, pravidelné konzultácie a trpezlivosť, ktorú so mnou mal. Taktiež by som chcela poďakovať rodine a kamarátom, ktorí ma podporovali a boli mi oporou v ťažkých chvíľach.

Obsah

Kapitola 1

Úvod

V dnešnej dobe možno nájsť množstvo rôznych hororových hier. Každá z nich ponúka inú atmosféru, príbeh a zážitok z hrania. Avšak väčšina z nich má jednu vec spoločnú – prostredie, príbeh a udalosti sú predom definované a naprogramované. Priebeh hry sa časom hráč naučí naspamäť a preto nemá po opätovnom hraní hry taký zážitok, ako po prvý krát.

Čo tak ale vytvoriť niečo nové? Hororovú hru, ktorá nebude ako žiadna iná. Miesto toho, aby hráč chodil cez predom definované prostredie, s predscriptovanými udalosťami a hral hru, tak by sa naopak ona hrala s hráčom ako s hračkou. Hra by mala byť nepredvídateľná, mala by sa stále meniť v závislosti od toho, ako sa hráč chová a mala by generovať náhodné udalosti a stretnutia v reálnom čase. Tým by hra bola zakaždým iná, zaujímavá, aj keby ju hráč hral po stý krát a prinášala by mu nové zážitky. Zároveň by v ňom stále vyvolávala pocit strachu, alebo minimálne zvedavosť z nepoznaného.

Cieľom tejto práce bolo vytvoriť herné demo v 3D, ktoré bude čo najviac procedurálne vytvárať a meniť prostredie a snažiť sa vyvolať v hráčovi záujem spojený so strachom na základe náhodne vygenerovaných a rozmiestnených udalostí.

Táto práca sa venuje popisu, návrhu a implementácie rôznych herných mechaník. Výsledok výslednej aplikácie možno vidieť v kapitole 2. Okrem výslednej aplikácie sú tu aj popísané hry, ktoré boli inšpiráciou pri vytváraní herného dema. Kapitola 3 sa zaoberá teóriou strachu, herného vývoja a hernými enginami, ktoré sú v dnešnej dobe populárne. Návrh jednotlivých herných mechaník a nápadov, ktoré vznikli počas vývoja, je popísaný v kapitole 4. Aby bolo možné lepšie pochopiť výslednú implementáciu, je v kapitole 5 popísaný herný engine Unity. V kapitole sú zmienené len základné funkcionality, ktoré boli použité pri vývoji. Výsledná implementácia herných mechaník je popísaná v kapitole 6. Na záver prebehlo ešte testovanie na užívateľoch, ktoré je popísané v kapitole 7.

Kapitola 2

Predstavenie hry a konkurenčné aplikácie

Táto kapitola predstavuje hry, ktoré boli inšpiráciou pre vytvorenie výslednej aplikácie. V ďalšom kroku je popísaná výsledná aplikácia.

2.1 Konkurenčné hry

Medzi veľmi populárne hry, z ktorých sa dá čerpať množstvo inšpiráciie a boli aj predlohou pre túto aplikáciu je Darkwood, Alan Wake, Visage, Pray a Cry of Fear. Každá z týchto hier ponúka iné zaujímavé herné mechaniky či už hororového charakteru alebo interakcie s nepriateľmi či prostredím.

Darkwood[1]

Hra sa zameriava na prežitie v temnom a tajomnom lese, kde sa nachádza hráčov hlavný hrdina. Hráč-hlavný hrdina musí prežiť niekoľko nocí v lese a vysporiadať sa s rôznymi nepriaznivými udalosťami a scénami, ktoré ho priamo ohrozujú (obrázok 2.1). Hra ponúka hráčom veľkú voľnosť v rozhodovaní, čo sa týka postupu a príbehu. Hráč sa môže rozhodnúť pre rôzne úlohy, ktoré mu prinesú nové možnosti, zdroje a informácie, vďaka ktorým prežije v lese. Okrem toho sa musí postarať o svoje zdravie, hygienu a hlad, čo dodáva hre prvok holého prežitia. Hra ponúka veľmi tajomnú a temnú atmosféru, ktorá hráčov udržuje v neustálom napätí. Hráč musí v noci strážiť svoje útočisko pred príšerami, ktoré sa snažia dostať dovnútra. Hra obsahuje aj prvok náhodnosti, takže hráč nikdy nevie, čo sa bude diať. Obsahuje prvky procedurálne generovaného obsahu, ktorý umožňuje náhodne generovať určité prvky ako je lokácia alebo nepriatelia. Tieto prvky pomáhajú udržiavať hru sviežu a zaujímavú.

Alan Wake[3]

Alan Wake je hororová akčná adventúra. Hlavným hrdinom je Alan Wake, úspešný spisovateľ hororových románov. Hra sa odohráva vo fiktívnom mestečku Bright Falls, kde sa stratí jeho žena Alice. Alan sa ju preto rozhodne hľadať a dostáva sa tak do temného a

[1] https://store.steampowered.com/app/274520/Darkwood/
[2] https://cultureofgaming.com/wp-content/uploads/2022/12/darkwood.jpg
[3] https://en.wikipedia.org/wiki/Alan_Wake

Obrázek 2.1: Obrázok z hry Darkwood[2]. Mapa v tejto hre je procedurálne generovaná na začiatku, pri spustení hry. Niektoré miesta sa generujú na náhodnej pozícií avšak niektoré sa objavujú vždy na rovnakých miestach.

tajomného sveta, ktorý sa podobá jeho vlastným románom. Hra obsahuje prvky psychologického hororu, ktoré sa prejavujú v podobe nejasných snov a videní, ktoré hlavná postava zažíva. Kľúčovým a inšpiratívnym prvkom tejto hry je osvetlenie (obrázok 2.2). Hráč musí využiť rôzne zdroje svetla, ako sú baterky, svetelné stĺpce, aby osvetlil temné prostredie a odhalil skrytých nepriateľov. Svetlo nepriateľov zraňuje a umožňuje hráčovi ich poraziť. Okrem osvetlenia nepriateľov sa svetlo používa aj ako obrana proti temným silám, ktoré sa vynárajú z tmy. Svetlo v tejto hre funguje nielen ako zbraň, ale aj k preskúmaniu prostredia. Hráč si musí dobre premyslieť, kedy použije ktoré zdroje svetla, a akým spôsobom bude s nimi pracovať a ktorých nepriateľov treba zneškodniť čo najskôr.

Obrázek 2.2: Obrázok z hry Alan Wake[4]. Hlavný nástrojom pre odhaľovanie bytostí a ich oslabenie je práve svetlo.

Visage[5]

Hra sa odohráva v temnej opustenej viktoriánskej vile. Hráč je ako postava, ktorá prišla o pamäť a snaží sa odhaliť pravdu o tom, čo sa v dome vlastne stalo. Hra sa skladá z niekoľkých kapitol, pričom každá má svoj vlastný jedinečný príbeh a atmosféru.

Jedným z najvýraznejších prvkov Visage je prostredie. Hra sa sústredí na silnú atmosféru, napätie, hrôzostrašné a tajomné prvky, ktoré sa objavujú v jednotlivých častiach vily. Hra obsahuje rôzne prvky hororu, ako sú jumpscares, neustále napätie a stresujúce situácie. Hráči sa môžu napríklad pohybovať po klaustrofobických chodbách, ktoré sú plné záhadných zvukov, výpadkov svetla a ďalších podobných desivých momentov (obrázok 2.3). Hra tiež používa rôzne ilúzie a halucinácie, ktoré majú hráča priviesť k tomu, aby pochyboval o tom čo vidí a čo je skutočné.

Ďalším zaujímavým prvkom je interaktivita s prostredím. Hráč môže v miestnostiach nájsť rôzne predmety, ktoré môže použiť k vyriešeniu záhad a ďalšiemu napredovaniu. Niektoré predmety môžu byť použité k otvoreniu skrytých miestností, alebo ku spusteniu niektorých mechanizmov.

Posledným výrazným prvkom sú puzzle. Hráč musí nazhromaždiť rôzne predmety a nájsť ukryté indície. Puzzle sú rôznorodé a často náročné a hráč preto musí použiť vlastnú inteligenciu, aby daný problém vyriešil.

[4]https://www.rockpapershotgun.com/alan-wake-remastered-review
[5]https://store.steampowered.com/app/594330/Visage/

Obrázek 2.3: Obrázok z hry Visage[6]. Visage sa vyznačuje veľkým množstvom jumpscarov.

Pray[7], [8]

Prey je sci–fi akčná hra s prvkami hororu. Hra sa odohráva na kozmickej stanici Talos I, ktorá krúži okolo Mesiaca. Hráči hrajú za postavu Morgana Yu, ktorý sa prebúdza na stanici bez spomienok na minulosť. Hráči musia prežiť a zistiť, čo sa stalo na stanici Talos I.

Jedným z najdôležitejších prvkov hry Prey je možnosť prechádzať medzi rôznymi formami. Morgana sa môže meniť na predmety, zvieratá a dokonca aj na mimozemské bytosti. Tento prvok umožňuje hráčom pristupovať k rôznym oblastiam a skrývať sa pred nepriateľmi.

Hra Prey je plná rôznych nepriateľov, ktorí sa menia a prispôsobujú sa hráčovmu štýlu hrania (obrázok 2.4). Hlavným nepriateľom sú Typhoni. Existuje niekoľko druhov Typhonov, pričom každý z nich má iné schopnosti. Nepriatelia zabezpečujú napínavú a strašidelnú atmosféru, pretože môžu útočiť z nečakanej polohy a ich schopnosti donútia hráča byť stále na pozore.

Najvýraznejším a najviac rozšíreným nepriateľom sú Mimici. Sú to malé tvory,ktoré majú schopnosť sa meniť na rôzne objekty v prostredí. Dokážu sa prerodiť do akéhokoľvek predmetu rovnakých alebo podobných rozmerov. Môžu sa premeniť na výbušniny, kancelárske stoličky a mnoho ďalších predmetov. Ich schopnosť maskovania im umožňuje hráča prekvapiť a zaútočiť v nečakaných situáciách. Mimici sú malé, rýchle a všestranné tvory. Ich schopnosť sa maskovať a imitovať okolité objekty im umožňuje vyvolávať zmätok a vystrašenie.

Ďalšími nepriateľmi sú napríklad Telepati, ktorí dokážu manipulovať s mysľou, Technopati, ktorí dokážu ovládať a manipulovať s elektronikou a mnohí ďalší.

[6] https://www.gamereactor.eu/video/322483/VisageOfficial+Gameplay+Trailer/
[7] https://store.steampowered.com/app/480490/Prey/
[8] https://segmentnext.com/prey-2017-enemies-guide/

Obrázek 2.4: Obrázok z hry Pray[9]. Na obrázku je možné rozpoznať nepriateľa "Mimics". Tento druh nepriateľov sa dokáže premeniť na rôzne predmety a útočiť na hráča z nepriateľskej polohy.

Cry of Fear [10]

Cry of Fear je hororová survival hra. Obsahuje viacero úrovní, ktoré sú rozdelené do kapitol. Hráči musia prejsť jednotlivými úrovňami, zbierať kľúče, riešiť záhady a bojovať proti rôznym nepriateľom. Cry of Fear má temnú a depresívnu atmosféru, ktorá je podporovaná hudbou a zvukovými efektami. Hra sa zameriava na hlavného hrdinu Simona Henrikssona, ktorý sa prebúdza v temnom a zvláštnom meste, ktoré je plné nebezpečenstiev.

Jedným z najzaujímavejších prvokov hry je použitie "psycho–logickej"mechaniky, ktorá umožňuje hráčom prechádzať do alternatívnych dimenzií, ktoré sa objavujú pri určitých udalostiach. Hráči musia využívať rôzne predmety na prekonanie nastrojených prekážok a aby sa mohli zorientovať v hre.

Cry of Fear ponúka aj veľa hádaniek, ktoré musia byť vyriešené, aby hráč mohol pokračovať v príbehu. Hráči si môžu pomôcť rôznymi nájdenými predmetmi, ktoré im umožnia postupovať ďalej v hre a odkryť viac z príbehu Simona.

Hra sa vyznačuje svojskou atmosférou, ktorá je plná napätia, hrôzy a neistoty. Tento strach sa zdôrazňuje pomocou temnej grafiky a dôrazu na zvukové efekty (obrázok 2.5). Hra má tiež viacero koncov, ktoré sa odomykajú v závislosti od rozhodnutí, ktoré hráči robia počas hry.

[9] https://www.youtube.com/watch?v=iUOv6RHrh9o
[10] https://en.wikipedia.org/wiki/Cry_of_Fear

Obrázek 2.5: Obrázok z hry Cry of fear[11]. Hráč sa nachádza na opustenej a temnej ulici, kde naňho číhajú rôzne hrôzostrašné nástrahy.

2.2 Výsledná aplikácia

Hráč sa prebúdza v temnom labyrinte, do ktorého sa záhadným spôsobom dostal a ostal v ňom uviaznutý. Nič nevie, nič si nepamätá, a preto sa chce dostať čo najrýchlejšie preč.

Jeho úlohou je nájsť východ a dostať sa z tohto labyrintu preč. Na to, aby bol schopný otvoriť východ, musí chodiť po labyrinte a nájsť všetky kľúče.

Aby to však nemal úplne jednoduché, číhajú naňho v labyrinte rôzne nástrahy a prekvapenia, ktoré ho majú vystrašiť. V hre sa stretáva s dvomi druhmi nepriateľov, ktorý ho ohrozujú a keď nie je dostatočne opatrný, môžu ho zabiť.

[11] http://pic.962.net/up/2012-2/201222984845986080.jpg

(a) (b)

Obrázek 2.6: Finálna podoba hry. Hra sa snaží, aby prostredie bolo temné a hráč sa cítil nepríjemne. Okrem toho, že v hre sa nachádzajú rôzne strašidelné predmety (obrázok a), číhajú naňho aj nepriatelia (obrázok b).

Kapitola 3

Strach a herní vývoj

Tato kapitola popisuje teoretický základ práce. Na začiatku je riešený koncept strachu. Za tým nasleduje časť, ktorá sa venuje hrám a hernému vývoju. Na konci sú predstavené najznámejšie herné enginy.

3.1 Strach

Wolf vo svojej knihe [16] hovorí, že strach označuje pocit, ktorý sprevádza akútny nepriaznivý stav, konkrétne ohrozenie. Je to stav, kedy je racionálne myslenie potlačené do úzadia a jedincovi dominujú inštinkty a naučené, nadobudnuté techniky, ktoré použije podvedomo-automaticky, bezmyšlienkovite. Strach vedie k nabudeniu organizmu vylučovaním podporných hormónov na lepšie telesné zvládnutie kritickej situácie, môže viesť k boju, úteku alebo apatii. Strach ukazuje, že jedinec je v stave aktuálneho ohrozenia a že ide nejako tento nepriaznivý stav riešiť, ochrániť sa, najlepšie bez osobnej ujmy.

Lenka Hanovská [4] uvádza, že strach, úzkosť a podobné fóbie sú často klasifikované ako internalizujúce problémy so správaním a sú pomerne bežné či už u detí alebo dospelých. Je to úzkostný stav, ktorý človek vníma vo svojom tele. Telesné symptómy sú spúšťané vegetatívnym nervovým systémom.

Strach možno tiež vnímať ako genetickú informáciu, ako uvádza kniha Jána Jelineka [6]. Po tisícročia sprevádzal ľudstvo a pomáhal mu vyhnúť sa reálnym nebezpečenstvám, ktoré v priebehu dejín ohrozovali jeho život a pomáha mu prežiť až po súčasnosť.

Podľa práce [12] prvé štúdie o strachu u mladých boli vykonané už z roku 1897. V jednej z týchto štúdií skúmali obavy u 1701 osôb vo veku 0 až 26 rokov. Medzi najčastejšie patrili búrky, plazy, cudzinci, tma, oheň, smrť, voľne žijúce zvieratá, choroby s priemerom 3,6 strachu na osobu.

Pocity strachu prichádzajú v rôznej sile a v rôznych situáciách. Rozlišujú sa dva základné druhy strachu:

- Akútny strach — objavuje sa náhle, počas niekoľkých dní. Jeho intenzita je premenlivá, avšak nikdy nezmizne. Prichádza vždy nečakane a človeka, ktorého postihne, často úplne ochromí. Záchvaty môžu trvať niekoľko minút no niekedy až niekoľko hodín a môžu prichádzať viackrát do týždňa či mesiaca. Väčšinou ide o strach z fyzickej ujmy alebo z nejakej „katastrofy“ v medziľudských vzťahoch [16].

- Chronický strach — rozvíja sa pomaly, opatrne mnohokrát až nepozorovane. Často je spojený s obavami z kritiky, odmietnutia a vlastného zlyhania [16].

Každý človek strach vníma inak a u každého sa prejavujú iné telesné symptómy. Typickým prejavom sú zvýšenie krvného tlaku, potenie, zmena prekrvenia pokožky, zrýchlený tlkot srdca, návaly tepla, pichanie v bruchu, či tlak v žalúdočnej oblasti. Svaly sú napnuté, kyslík z pľúc sa vytráca a človek nadobúda pocit, ako keby ho niekto spútal. Často sa môžu začať objavovať aj mdloby a nevoľnosť. Strach ovplyvňuje aj myšlienkové pochody. Sprevádza ho nízka schopnosť sústrediť sa a dochádza aj k výpadkom pamäte. Môžu sa začať objavovať nočné mory [6].

Strach sa môže priblížiť do podvedomia z rôznych dôvodov. Vždy je však spúšťačom ľudská myseľ. Je to veľmi silný nástroj, ktorý umožňuje myslenie, tvorbu predstáv a vedomie. Myseľ a jej produkty, myšlienky a predstavy sú pôvodcom emócií. Príčinou strachu sú teda vnútorné interpretácie, hodnotenia určitých udalostí a ich výsledné predpovede, ktoré si vedome, či nevedome v duchu človek sám sebe predkladá [6].

3.2 Hry, herní vývoj, herní enginy

Táto sekcia sa zaoberá teóriou hier a procesu ich vývoja. Venuje sa detailnejšiemu opisu hororových hier a mechaník, ktoré tieto hry využívajú. Vysvetľuje, čo je to pojem herný engine a k čomu ho vývojári pri tvorbe hier využívajú.

Hra

V literatúre možno nájsť mnoho definícií pre pojem hra. Kniha Rules of Play [2] uvádza, že vo všeobecnosti možno hru vnímať ako interaktívnu formu zábavy alebo činnosti, ktorá zahŕňa účasť hráča s cieľom dosiahnuť zábavu, rekreáciu, vzdelávanie alebo iné formy uspokojenia. Hry sa hrávajú za rôznych okolností a môžu byť fyzické (napríklad športy), mentálne (napríklad šach) alebo digitálne (počítačové hry). Hry často zahŕňajú pravidlá, ciele, interakciu medzi hráčmi a zážitok z dosiahnutia výsledkov.

Hra je aktivita pri ktorej sa rieši nejaký problém ku ktorému sa pristupuje s hravým postojom.

Hororové hry

Hororové hry sú videohry, ktoré sa zameriavajú na vyvolávanie strachu, napätia a neistoty u hráčov. Hráči sa zvyčajne ocitajú v nebezpečných a neznámych prostrediach, kde musia prekonať prekážky a hrozby, aby prežili.

Hororové hry sa zvyčajne vyznačujú temnou a depresívnou atmosférou, ktorá môže obsahovať rôzne prvky ako sú napríklad tichá hudba, nejasné obrazy, zvukové efekty, nečakané zvraty alebo strašidelné zvuky. Tieto prvky majú za cieľ vyvolať emócie hráča a prinútiť ho cítiť sa nepríjemne, zraniteľne a v ohrození. Niektoré z týchto prvkov, ako uvádza Bernard Perron [11], zahŕňajú:

- Jumpscares – Jumpscares sú neočakávané momenty, kedy sa v hre objaví niečo strašidelné a hráč sa vystraší. Môžu to byť napríklad zvuky, animácie alebo objekty, ktoré sa náhle a nečakane objavia na obrazovke.
- Temná a tajomná atmosféra – Hororové hry často vytvárajú temnú a tajomnú atmosféru, ktorá hráča udržuje v neistote. Táto atmosféra môže byť vytvorená pomocou špecifického svetla, zvukov a grafických prvkov.

- Nepríjemné zvukové efekty – Zvukové efekty sú kľúčovým prvkom v hororových hrách. Môžu to byť napríklad temné a hrozivé hudobné skladby, alebo zvuky, ktoré vzbudzujú nepríjemné pocity, ako napríklad šepkanie, chrčanie alebo kroky.
- Náročné puzzle – Hororové hry často obsahujú náročné úlohy, zvané puzzle, ktoré hráč musí riešiť, aby sa dostal ďalej. Tieto puzzle môžu byť súčasťou príbehu alebo prostredia, v ktorom sa hráč nachádza.
- Pocit beznádeje – Hororové hry môžu vyvolať pocit beznádeje u hráčov tým, že im dávajú len obmedzené možnosti prežitia v hre. Hráči sa môžu cítiť osamelí a zraniteľní, keď sa snažia prekonať prekážky a hrozby v hre.

Hororové hry sa stávajú stále populárnejšie medzi hráčmi a môžu byť zdrojom zábavy pre tých, ktorí milujú napätie a adrenalín. Avšak u citlivejších hráčov, môžu vyvolať tak intenzívny zážitok zo strachu, ktorý môže v nich vyvolať rôzne psychické problémy ako napríklad problémy so spánkom. Preto je dôležité zvážiť vlastnú toleranciu na hororové prvky pred hraním týchto hier. Viac o hororových hrách je možné sa dočítať v publikácií *Targeting horror via level and soundscape generation* [9].

Procedurálne generovanie

Autor knihy Procedural Generation in Game Desing [15] uvádza, že procedurálne generovanie je spôsob tvorby digitálneho obsahu pomocou matematických pravidiel a algoritmov, ktoré určujú jeho vlastnosti a vzhľad. Možno tak dosiahnuť vysokú úroveň variability a jedinečnosti.

V hernom priemysle je cieľom procedurálneho generovania vytvoriť hry s obrovským množstvom rôznych možností. Generovaním herných levelov, postáv a predmetov sa môžu hry odlišovať od seba a zároveň hráčovi prinášajú nové dojmy aj po opakovanom hraní hry.

Okrem herného priemyslu je možné procedurálne generovanie využiť aj v architektúre a urbanizme na tvorbu plánov, modelov budov a mestských oblastí. V medicíne sa procedurálne generovanie používa na tvorbu rôznych typov vizualizácií, simulácií a modelov. V umení je cieľom procedurálneho generovania vytvoriť jedinečné a originálne umelecké diela pomocou počítačových algoritmov a programov. O procedurálnom generovaní v hrách je možné dočítať sa viac v publikácií *Sonancia: Sonification of procedurally generated game levels* [8].

Herný vývoj

Ako uvádza kniha The Art Of Game Desing [14], herný vývoj je proces tvorby a vývoja videohier. Tento proces sa zvyčajne skladá z niekoľkých fáz, ktoré zahŕňajú plánovanie, návrh, programovanie, testovanie a vydanie hry.

Herný vývoj môže trvať rôzne dlho, od niekoľkých mesiacov až po niekoľko rokov v závislosti od veľkosti tímu, rozpočtu a zložitosti hry. Veľké herné spoločnosti zvyčajne zamestnávajú tímy o stovkách ľudí, ktoré sa zameriavajú na rôzne aspekty herného vývoja, zatiaľ čo menšie tímy môžu byť zložené iba z niekoľkých ľudí, ktorí pracujú na menších projektoch.

Plánovanie Prvou fázou je plánovanie. V tejto fáze tímy tvorcov hier určujú, aký typ hry chcú vytvoriť, aký je jej cieľový trh, aké sú rozpočty a aké sú požiadavky na zdroje a časový harmonogram.

Proces plánovania je zložený z niekoľkých častí – vytvorenie konceptu hry, stanovenie cieľov projektu, vytvorenie projektového plánu, zabezpečenie financií, stanovenie role tímu, rozvrhnutie úloh, testovanie a kontrola plánu.

Najkritickejším prvkom procesu plánovania je testovanie. Je potrebné stanoviť testovacie plány a stanoviť, kedy sa bude testovanie realizovať a aké testovacie metódy sa použijú [14].

Návrh Podľa knihy, na ktorej sa podieľal Jim Thompson [7], sa fáza návrhu pri vývoji hier zameriava na tvorbu konceptu a koncepcie hry, jej herných mechaník, vizuálnych prvkov a príbehu. Cieľom tejto fázy je vytvoriť základný plán hry a zabezpečiť, aby herná skúsenosť bola zábavná, intuitívna a pre hráčov atraktívna. Vývojové tímy spoločne vytvárajú celý herný koncept, zdieľajú svoje myšlienky a nápady a tvoria vizuálne prvky hry.

Podobne ako proces plánovania, tak aj návrh je zložený z niekoľkých krokov – navrhnutie herných mechaník, navrhnutie vizuálnych prvkov, navrhnutie príbehu, vytvorenie prototypu a testovanie.

Prototypy sa používajú na testovanie a vylepšovanie konceptu a koncepcie hry, ako uvádza vo svojej knihe Kevin Saunders [13]. Prototypy môžu byť jednoduché skice, alebo dokonca funkčné hry, ktoré sa používajú na testovanie herných mechaník a vizuálnych prvkov.

Testovanie sa používa na odhalenie chýb a nedostatkov v koncepte a koncepcii hry a pomáha zabezpečiť, aby sa hra v budúcnosti vyvíjala správnym smerom.

Programovanie V tejto fáze sa už implementujú a vytvárajú herné mechanizmy a interakcie, ktoré boli vytvorené v predchádzajúcich fázach. Programátori používajú rôzne programovacie jazyky a nástroje, ako sú herné engine a nástroje pre vizuálny dizajn, na implementáciu a testovanie herných mechanizmov [7].

Najpoužívanejšími programovacími jazykmi sú C++, C#, Java alebo Python. Okrem toho sa používajú aj herné enginy, ktoré poskytujú programátorom rôzne funkcionality a nástroje pre implementáciu herných mechanizmov a interakcií.

Veľké herné spoločnosti majú k dispozicií svoje vlastné herné enginy, ktorých implementácia je prispôsobená tak, aby sa vývojárom hra dobre a jednoducho vytvárala.

Testovanie Fáza testovania pri vývoji hier sa zameriava na odhalenie chýb, problémov a nedostatkov v hre pred jej uvedením na trh. Testovanie by malo byť dôkladné a zamerané na rôzne aspekty hry, vrátane funkčnosti, výkonu, hrateľnosti, použiteľnosti, grafiky a zvukových efektov [14].

Pri testovaní sa používajú rôzne techniky, ako sú manuálne testovanie, automatizované testovanie a testovanie kompatibility na rôznych platformách. Okrem toho sa používajú aj rôzne nástroje a softvérové riešenia na zlepšenie a zjednodušenie testovacieho procesu.

Vydanie Fáza vydania pri vývoji počítačových hier zahŕňa uvedenie hry na trh a jej distribúciu hráčom. Táto fáza je kľúčová pre úspech hry, pretože vydanie hry na trh je často dlho očakávaným momentom pre fanúšikov a zákazníkov [14].

Pri vydávaní hry je dôležité zvoliť správny čas a miesto na jej uvedenie na trh. Zvyčajne sa hry vydávajú cez digitálne platformy ako Steam, GOG, Epic Games Store alebo PlayStation Store, ale môžu sa tiež predávať na fyzických médiách ako CD alebo DVD. Okrem toho môže byť hra distribuovaná aj cez mobilné platformy ako App Store alebo Google Play.

V rámci tejto fázy musí byť zabezpečená aj marketingová kampaň, ktorá zvýši povedomie o hre a priláka potenciálnych zákazníkov. Marketingová kampaň môže zahŕňať rôzne aktivity, ako sú propagačné videá, články v herných médiách, súťaže alebo interaktívne kampane [13].

Okrem toho je potrebné zabezpečiť aj technickú podporu a správu vydania hry. V prípade, že sa objavia chyby alebo problémy, musí byť rýchlo poskytnutá oprava a aktualizácia hry.

Herné enginy

Ako popisuje kniha Game Engine Architecture [3], herné enginy sú softvérové nástroje určené na vývoj videohier a interaktívneho obsahu. Sú navrhnuté tak, aby zjednodušili a zrýchľovali vývoj hier, umožnili tvorbu hier bez rozsiahlej znalosti programovania a poskytli programátorom a dizajnérom nástroje potrebné na tvorbu vysokej kvality hry.

Vývojári s ich pomocou môžu vytvárať a zobrazovať 3D svety a objekty, ktoré sa správajú v súlade s fyzikálnymi zákonmi. Každý herný engine obsahuje systém vykresľovania grafiky, ktorý dokáže zobrazovať 3D a 2D objekty a animácie, rôzne vizuálne efekty, osvetlenie a ďalšie prvky. Pomocou fyzikálnych zákonov dokážu simulovať pohyb a kolízie objektov, gravitáciu, vzduchový odpor a iné fyzikálne vlastnosti. Zabezpečujú sa tak správne pohyby a odozvy objektov a ich interakcie s prostredím a inými objektami. Herné enginy umožňujú pridať zvukové efekty a hudbu, ktoré prispievajú k tvorbe atmosféry v hre. Okrem toho sa s nimi dajú vytvárať animácie, ktoré dodávajú postavám a objektom v hre pohyblivosť a životnosť. Herné enginy poskytujú možnosť písania skriptov, ktoré umožňujú programátorom vytvárať interakciu s objektami v hre. Užívateľské rozhranie, ktoré herné enginy poskytujú, slúži na tvorbu menu, nastavení a scén. Navyše, herné enginy sú schopné tvoriť hry pre rôzne platformy vrátane PC, mobilných zariadení, konzol a ďalších [3].

Niektoré herné enginy poskytujú aj nástroje na vytváranie umelej inteligencie pre NPC postavy a vizuálne skriptovacie nástroje, ktoré zjednodušujú tvorbu logiky hry bez potreby programovania.

Medzi najpopulárnejšie herné enginy na trhu patria Unreal Engine, Unity, CryEngine, Godot Engine a mnoho ďalších. Každý engine má svoje vlastné výhody a nevýhody a určený je pre rôzne typy hier a platformy.

Výhodou použitia herných enginov je, že vývojári nemusia vytvárať všetky funkcie a nástroje od základu, čo môže ušetriť veľa času a úsilia. Vývojári môžu tiež využívať existujúce komunity a zdroje pre herné enginy, čo im umožňuje rýchlejšie učenie sa a rýchlejší vývoj hier [1].

[1] https://www.g2.com/categories/game-engine

Kapitola 4

Návrh

Táto kapitola obsahuje návrh rôznych herných mechaník a nápadov, ktoré vznikali počas vývoja. Na začiatku sú rozobrané problémy dnešných hororových hier a návrhy, ako tieto problémy zlepšiť. Ďalej je v kapitole je popísaný spôsob, ako niektoré návrhy realizovať.

4.1 Problém konkurenčných hier a ich riešenie

Ako bolo spomenuté, väčšina dnešných hier je založená na predom definovaných udalostiach. Aj keď sa v niektorých vyskytujú nejaké procedurálne prvky, stále je väčšina z nich postavená na predscriptovaných udalostiach a prostredí.

4.1.1 Jumpscares

Sú síce jedným zo základných prvkov každej hororovej hry, avšak vzhľadom na ich skriptovateľnosť poškodzujú znovuhrateľnosť hry. Keď ich hráč v hre zažije, vie presne kde a kedy sa vyskytujú. Vďaka tomu už opätovný prechod hrou nie je zďaleka taký intenzívny ako ten prvý.

Riešenie vopred naskriptovaných jumpscares spočíva v náhodne generovaných stretnutiach/udalostiach na základe vopred definovaných kritérií. Tieto akcie nebudú obmedzené na konkrétne oblasti, kde sa môžu spustiť – môžu sa stať úplne kdekoľvek a kedykoľvek, pokiaľ sú splnené spúšťacie kritériá. Cieľom je urobiť hru čo možno najviac nepredvídateľnou a viazanou na akcie hráča.

Každá udalosť/stretnutie bude mať definované konkrétne podmienky, ktoré je potrebné splniť, aby sa tak stala spustiteľnou. Tieto podmienky môžu napríklad zahŕňať:

- aktuálna poloha hráča (dlhá úzka chodba, tmavá miestnosť atď.)
- blízkosť konkrétnych objektov, s ktorými možno interagovať alebo ich použiť na boj (svetlá ktoré môžu začať blikať, televízor alebo rádio, ktoré sa môže náhodne zapnúť atď.)
- momentálna vizuálna scéna (napr. vlastný odraz v zrkadle s niečím za jeho chrbtom a po otočení to „niečo“ tam môže alebo nemusí byť)
- ďalšie premenné prepojené s konkrétnou činnosťou, alebo osobou (napr. či má hráč zapnutú baterku alebo nie, atď.)

Množstvo udalostí musí byť čo najširšie. Môže to byť len niečo malé, napríklad náhle blikajúce svetlo na niekoľko sekúnd alebo dvere, ktoré sa môžu pomaly otvárať za sprievodu škrípavého zvuku, až po stretnutie s jedným alebo viacerými nepriateľmi.

4.1.2 Creepiness vs scariness

Veľa hororových hier buduje svoju atmosféru na násilí, krvi, rušivom prostredí a nepriateľoch, na ktorých je nepríjemné sa vôbec len pozrieť. Môže sa potom stať, že hráč je po čase z prostredia skôr znechutený než vystrašený. Hra by nemala v hráčovi vyvolávať znechutenie, ale mala by ho donútiť cítiť sa nekomfortne, nervózne, zraniteľne. Hráč by mal byť vystrašený z akéhokoľvek zvuku, tieňa či pohybu.

4.1.3 Zvuky

Zvuky na pozadí sú síce veľmi dôležitou súčasťou pri budovaní atmosféry, avšak hráč sa časom naučí, že množstvo zvukov, ktoré počuje, je neškodných a neznamenajú absolútne nič. Takže miesto toho, aby sa použili na pozadí dodatkové zvuky, budú sa náhodné spúšťať rovnaké, aké môže vydávať nepriateľ (vrčanie, škrabanie, praskanie skla atď.). Môže to potom znamenať, že nepriateľ môže byť za rohom, ale aj nemusí.

4.1.4 Nepriatelia a súboj

Za predpokladu, že hráč hral špecifickú hororovú hru už niekoľko krát, pozná všetky udalosti, jumpscares a zvukové efekty, neostáva už skoro nič, čo by ho mohlo prekvapiť a čo by v ňom mohlo vyvolať nejaké emócie. Pre takéhoto hráča sa hra stáva nudnou, rutinnou – iba kráčanie, beh a strieľanie. Väčšina hier preto, aby aspoň trochu zdvihla napätie, dá hráčovi len malé množstvo munície a životov. Tým sa zvýši hráčova zraniteľnosť. Tento spôsob môže byť síce efektívny, ale nie je to ideálne riešenie. Myšlienkou je, že miesto toho aby sa len znižovalo množstvo munície, tak by pre každého nepriateľa existoval iný spôsob, ako s ním bojovať. Niektorý nepriatelia by boli zraniteľný pomocou vody, iný za pomoci elektriny a iných by dokonca mohli vystrašiť špecifické zvuky. Takže namiesto toho, aby hráč len klikal na myš, bude musieť zvoliť správny spôsob, ako s nepriateľom bojovať. Môže to buď za pomoci predmetov, uložených v inventári alebo za pomoci prostredia (visiace elektrické káble). Niektorých nepriateľov by dokonca nebolo možné zabiť, len oslabiť na nejaký čas, prípadne odstrašiť. To by však znamenalo, že nepriateľ je stále niekde nablízku, môže sa vrátiť a znovu začať útočiť.

4.1.5 Procedurálne generované a meniace sa prostredie

Problém procedurálne generovaného prostredia je ten, že väčšinou pozostáva zo skupiny predom definovaných modulov, ktoré sa spájajú a vytvárajú herný svet. Vo výsledku prostredie často vyzerá nevýrazne, opakuje sa a vyzerá horšie v porovnaní so statickým prostredím. Myšlienkou je, generovať prostredie iba v špecifických prípadoch. Hráč sa napríklad nachádza v budove, pohybuje sa cez izby a chodby pomocou otvárania dverí. Budova nemusí byť statická, ale môže sa vždy jej rozloženie meniť v závislosti od pohybu hráča. Príkladom môže byť rovná chodba. Hráč dôjde na jej koniec a tam nie je nič len stena. Tak sa rozhodne otočiť a ísť naspäť. Avšak pred ním sa objavia dvere do miestnosti, ktorá je celá krvavá a plná mŕtvych tiel. Tento princíp by mohol byť použitý aj na uväznenie hráča v špecifickej miestnosti, kým nevyrieši hádanku. Izba by mala niekoľko dverí. Keď prejde niektorými z

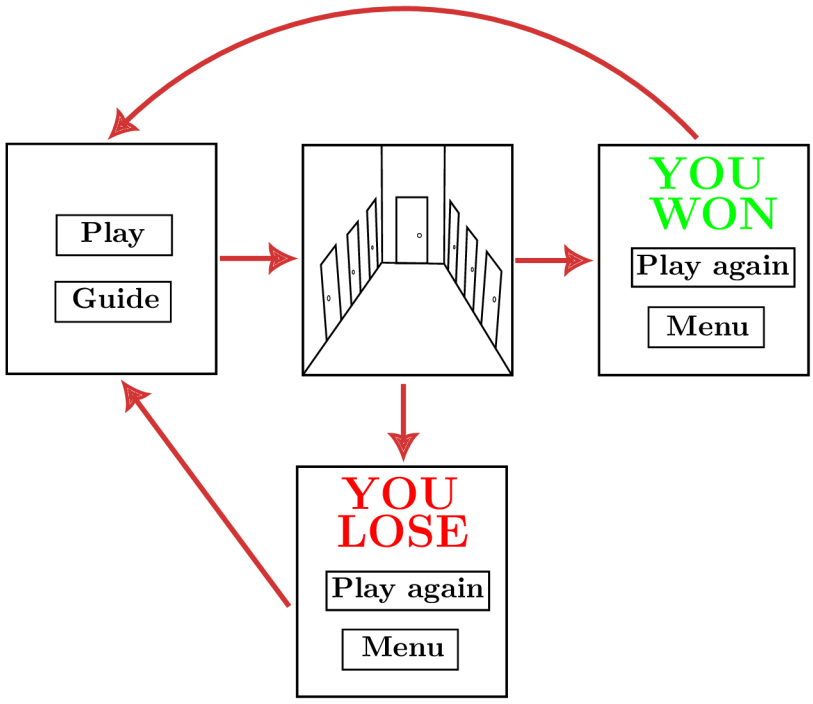
nich, vždy sa objaví v pôvodnej miestnosti. Až po vyriešení hádanky by všetky dvere zmizli a zobrazili by sa len jedny, cez ktoré by bolo možné uniknúť von.

4.2 Výsledné demo

Obsiahnuť takéto veľké množstvo nápadov a generovania by zahŕňalo vývoj na niekoľko rokov a preto cieľom tejto práce je vytvoriť herné demo, ktoré zahrnie aspoň pár týchto bodov. Práca sa zameriava na vygenerovanie náhodného labyrintu, cez ktorý môže hráč prechádzať. Jeho cieľom je nájsť dvere, ktoré vedú von a vyhľadať kľúče, ktoré odomknú tieto dvere a uniknúť tak z labyrintu. Aby to nemal jednoduché, sú v hre rozmiestnení nepriatelia, ktorí mu uberajú životy a môžu ho dokonca aj zabiť. Atmosféra hry je postavená na temnom prostredí a tichu. Ticho prerušuje len ozvena chôdze hráča a občasné náhodné zvuky, ktoré môžu, ale aj nemusia znamenať príchod nepriateľa alebo jumpscaru.

4.3 Životný cyklus hry

Hráč sa v priebehu hrania stretáva so 4 hernými pohľadmi – úvodným menu, gameover menu, koncový menu a s pohľadom, kde sa odohráva hlavný dej hry. Všetky medzi sebou plynule prechádzajú (obrázok 4.1).



Obrázek 4.1: Prechod medzi jednotlivými hernými pohľadmi

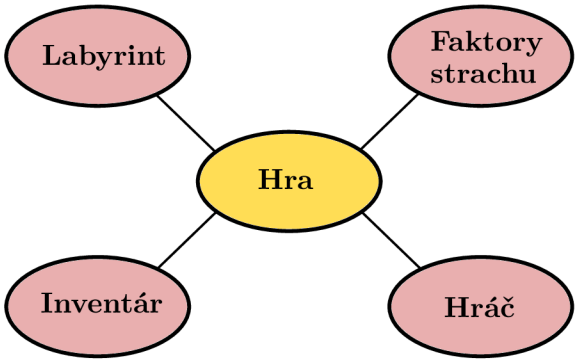
4.3.1 Úvodné a koncové menu

Po zapnutí hry hráča uvíta jednoduché menu s dvomi tlačidlami. Jedno umožní hru spustiť. Pomocou druhého si hráč môže zobraziť návod, ako hru ovládať. Keď hráčov život klesne na 0, objaví sa gameover obrazovka. Hráč môže konkrétnu hru začať hrať od znova, alebo sa môže vrátiť do úvodného menu.

Po zozbieraní všetkých kľúčov a otvorení únikových dverí hráč vyhral a má možnosť sa buď vrátiť do menu alebo si vygenerovať nový level.

4.3.2 Herný pohľad

Hlavnou a najdôležitejšou časťou, kde sa všetko vykonáva a generuje, je samotný herný pohľad. Ten je zložený z miestností, hráča, inventára a faktorov strachu (obrázok 4.2).

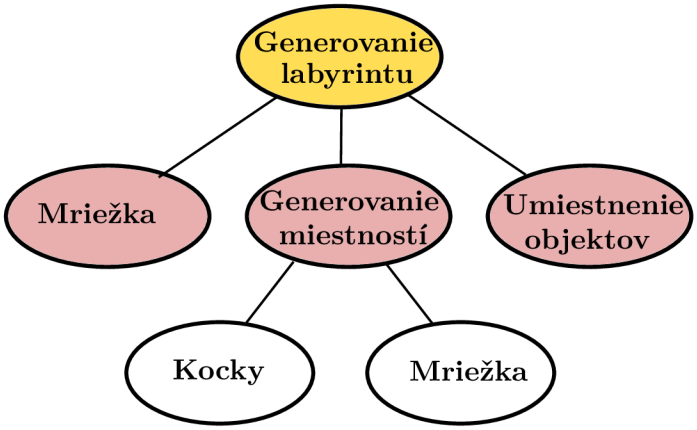


Obrázek 4.2: Základné prvky, z ktorých je zložená hlavná herná scéna.

Ako prvé sa vygenerujú jednotlivé miestnosti. Po ich vygenerovaní sa môžu rozmiestniť predmety, faktory strachu a hráč. Hráč sa pohybuje po jednotlivých miestnostiach, pracuje s inventárom, zbiera predmety a interaguje s jednotlivými faktormi strachu.

4.4 Generovanie labyrintu

Generovanie labyrintu je zložené z 3 fáz - vytvorenie mriežky, izby a následného rozmiestnenia objektov (obrázok 4.3).



Obrázek 4.3: S generovaním labyrintu súvisia pojmy ako sú mriežka, generovanie miestností a umiestnenie objektov. Ako prvé sa vytvorí mriežka, na ktorú sa budú jednotlivé miestnosti ukladať. Miestnosti sú zložené z kociek, ktoré sa taktiež umiestňujú do mriežky. Po vygenerovaní miestností je možné umiestniť objekty.

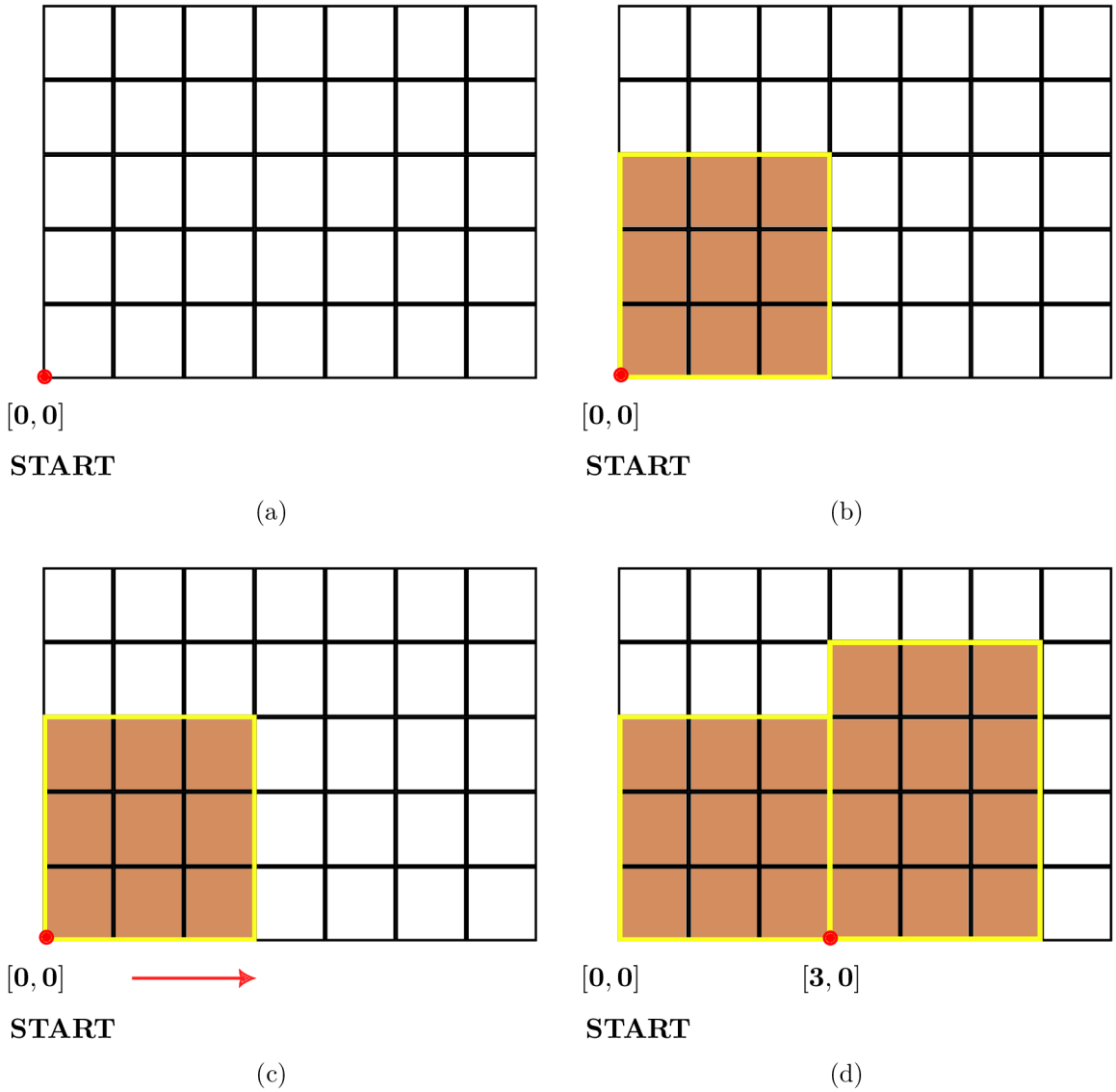
Pred samotným generovaním je najskôr potrebné vytvoriť mriežku, na ktorú sa budú ukladať vytvorené izby. Algoritmus následne prehľadáva jednotlivé bunky mriežky v ná-

hodne zvolenom poradí a zisťuje, či ešte niekde v mriežke existuje priestor, kde možno vytvoriť miestnosť. Postupne počas prechádzania jednotlivými bunkami sa vytvárajú izby, ktoré sa na tieto bunky ukladajú. Keď je celý labyrint vytvorený, môžu sa do jednotlivých izieb umiestniť objekty.

4.4.1 Generovanie miestností vedľa seba

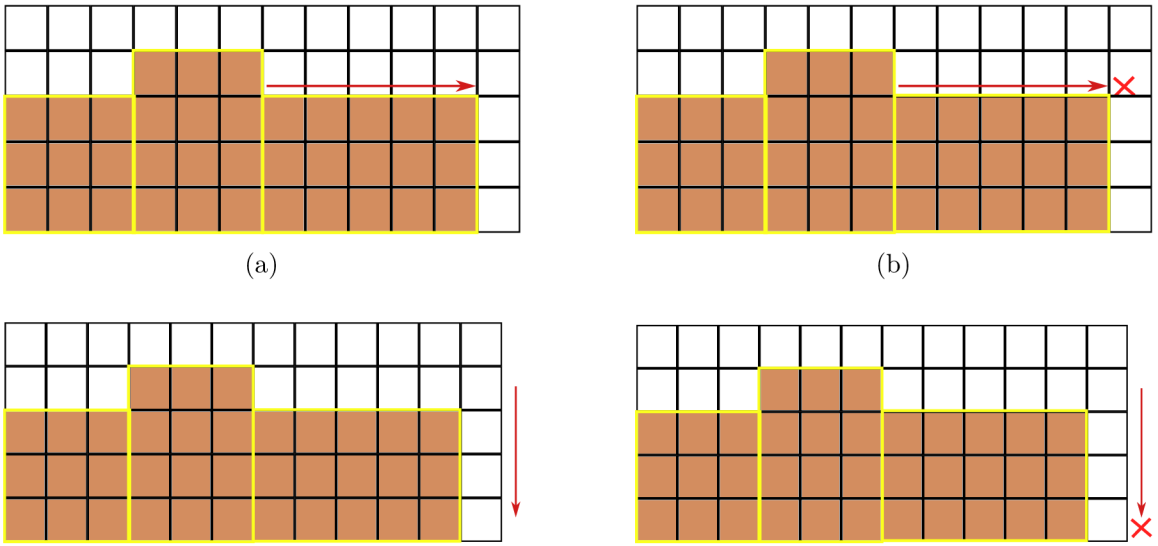
Základný algoritmus postupne prechádza mriežkou a ukladá na ňu vedľa seba miestnosti. Ako začiatočný bod na mriežke sa zvolí $[0, 0]$. Kvôli prirodzenejšiemu vzhľadu je minimálna veľkosť jednej izby $2m \times 2m$ a maximálna 10% z celkovej veľkosti mriežky.

Vygeneruje sa náhodná šírka a výška, podľa ktorej sa vygeneruje izba. Izba sa následne umiestni na mriežku (obrázky 4.4a a 4.4b). Bunky, ktoré jednotlivé kocky izby prekrývajú sa označia ako obsadené. Podľa toho bude algoritmus potom rozlišovať, či sa na dané miesto dá umiestniť izba alebo nie. Po umiestení izby sa náhodne vyberie smer, do ktorého sa pôjde generovať ďalšia izba. Bod štartu sa podľa toho posunie doľava, doprava, hore alebo dole a vygeneruje sa rovnakým procesom nová izba (obrázky 4.4c a 4.4d). Po posunutí sa môže proces generovania izby znovu opakovať, až kým nebude možné nikde na mriežku umiestniť izbu.



Obrázek 4.4: Postupné ukladanie izieb do mriežky. Ako prvé sa vytvorí prázdna mriežka, na ktorú sa budú postupne ukladať jednotlivé izby. Ukladanie sa začína od pozície štart. (obrázok a). Keď je izba vygenerovaná, umiestni sa na mriežku (obrázok b). Po umiestnení izby sa zvolí smer, kde sa bude generovať nová izba a začiatočný bod sa posunie (obrázok c). Následne sa na nových súradniciach vygeneruje nová izba (obrázok d).

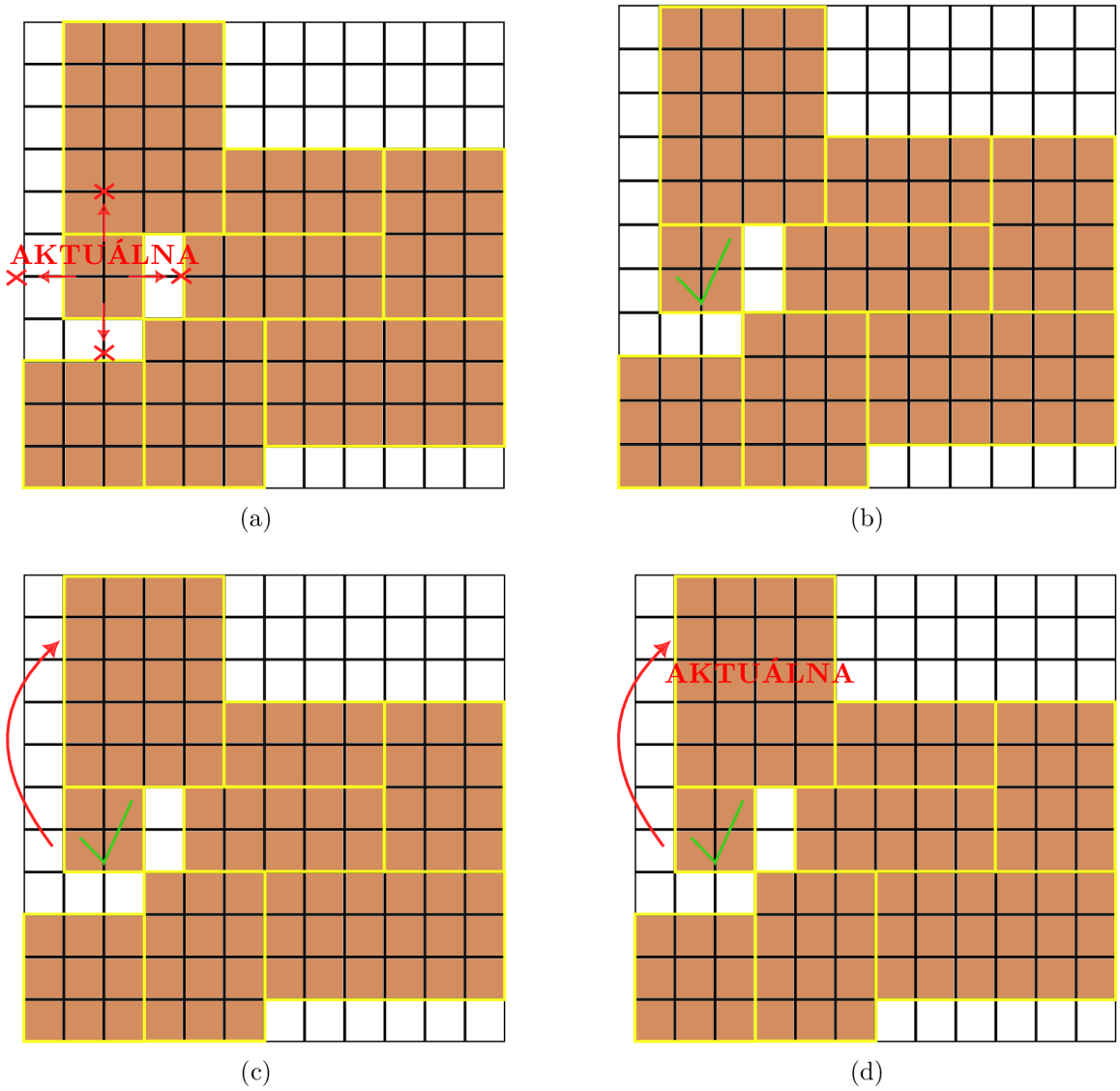
Pred samotným umiestnením izby je potrebné skontrolovať, či sa tam vôbec zmestí, alebo či nepresiahla hranice mriežky. Ak áno, skúsia sa zmenšiť jej rozmery. Pokiaľ nepomôže ani to, prehľadá sa okolie izby a skontroluje sa, či je vôbec možné vedľa aktuálnej izby umiestniť miestnosť (obrázok 4.5). Pokiaľ nie, algoritmus sa vráti k predchádzajúcej izbe a skúsi miestnosť vygenerovať vedľa nej (obrázok 4.6). Hlavá myšlienka algoritmu rozmiestňovania miestností je založení na backtrackingu, konkrétne na algoritme Depth-First Search.



(a) (b)

(c) . (d)

Obrázek 4.5: Pokus o umiestnenie izby v jednotlivých smeroch. Algoritmus vyberie nový smer pre generovanie do ľavej strany od pozície aktuálnej izby (obrázok a). V smere doľava nie je možné umiestniť izbu, pretože sa tam nenachádza dostatočný počet buniek, kde by sa mohla vygenerovať izba. Tento smer je preto neplatný (obrázok b). Algoritmus teda vyberie nový smer pre generovanie nadol od pozície aktuálnej izby (obrázok c). V smere nadol nie je možné umiestniť izbu, pretože by prekročila hranice mriežky. Tento smer je preto neplatný (obrázok d).

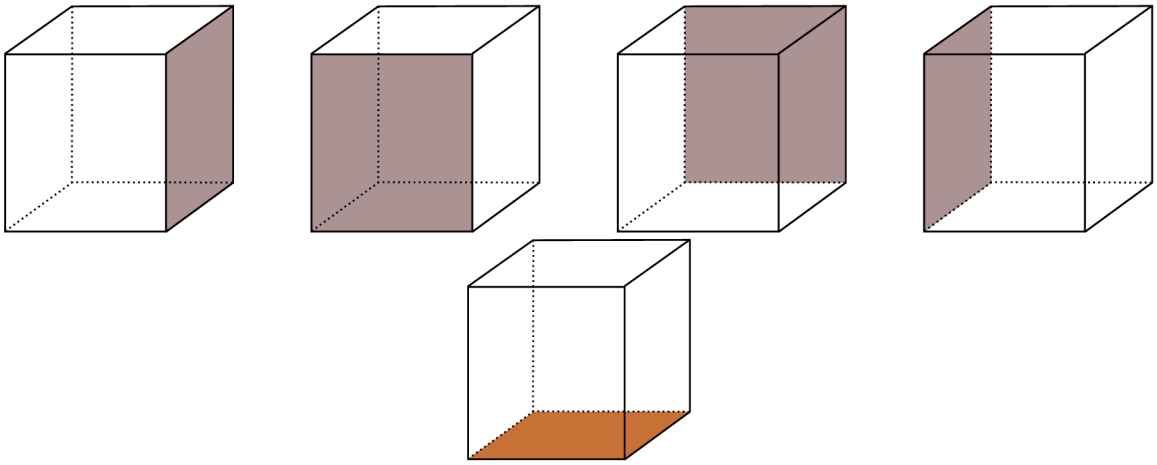


(a) (b) (c) (d)

Obrázek 4.6: Princíp backtrackingu pri umiestňovaní izieb. Okolo aktuálne spracovávanej izby nie je žiadny priestor, kde by sa mohla vytvoriť nová izba (obrázok a). Izba sa preto označí ako spracovaná (obrázok b). Algoritmus sa posunie na predchádzajúcu izbu (obrázok c). Tá sa stane aktuálnou a algoritmus sa pokúsi novú izbu vygenerovať vedľa nej (obrázok d).

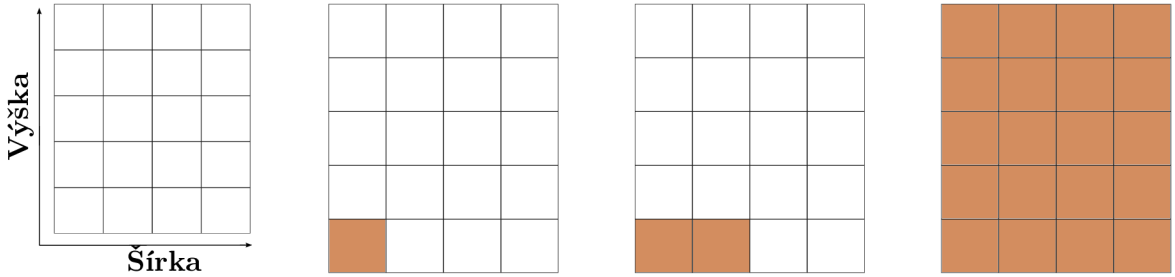
4.4.2 Generovanie izby

Základnou jednotkou pre generovanie izby je kocka o rozmeroch $2m \times 2m \times 2.5m$. Algoritmus pracuje s piatimi druhmi kociek – kocka s ľavou, pravou, dolnou, spodnou stenou a dlážkou (obrázok 4.7).



Obrázek 4.7: Druhy kociek, z ktorých sú zostavené jednotlivé miestnosti.

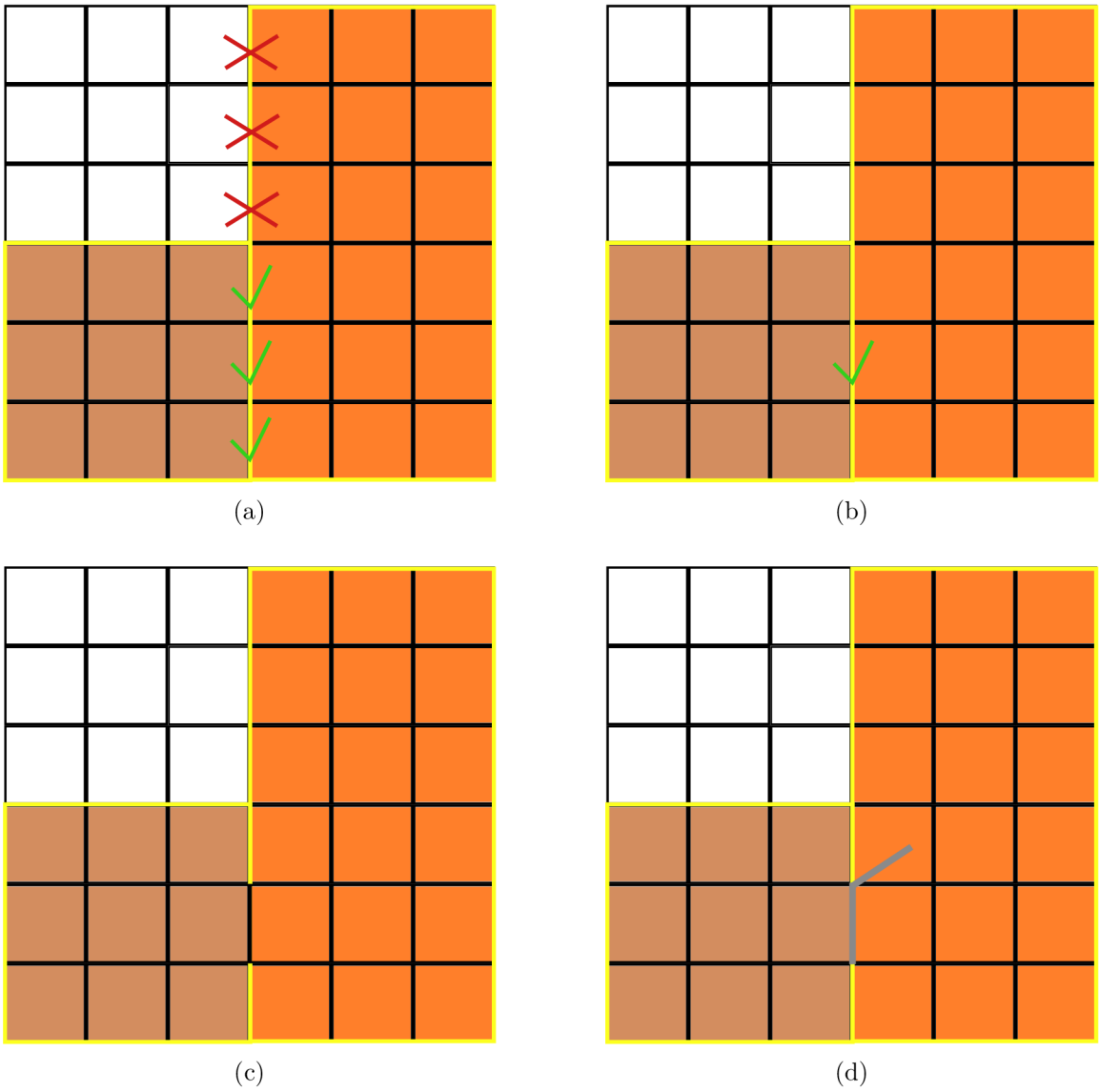
Jednotlivé kocky sa umiestňujú do mriežky, ktorá sa vytvára na základe náhodne vygenerovanej šírky a výšky. Cez mriežku sa prechádza bunka po bunke, riadok po riadku. Podľa aktuálnej polohy v mriežke sa na aktuálnu bunku umiestni príslušná kocka - obrázok 4.8.



Obrázek 4.8: Postup vytvárania izby. Ako prvé sa vygeneruje náhodná šírka a výška. Následne sa postupne do mriežky ukladajú jednotlivé kocky a vznikne celá izba.

4.4.3 Generovanie dverí

Keď je miestnosť vygenerovaná, nasleduje fáza umiestnenia dverí. Skontrolujú sa bunky, ktoré sú spoločné medzi novo vygenerovanou izbou a izbou, vedľa ktorej sa generovala (obrázok 4.9a). Ak sú dve bunky vedľa seba obsadené, znamená to že ide o stenu, ktorú majú spoločnú. Takéto bunky sa pridajú do zoznamu spoločných buniek a následne sa náhodne vyberie jedna, kde sa umiestnia dvere (obrázok 4.9b). Steny na týchto bunkách budú zmazané a nahradené dverami, ktoré umožnia hráčovi prejsť medzi izbami (obrázky 4.9c a 4.9d).



Obrázek 4.9: Nahradenie náhodnej spoločnej steny dverami. Bunky medzi novou a starou izbou sa skontrolujú. Tie, ktorú sú spoločné sa označia (obrázok a). Zo zoznamu spoločných buniek sa náhodne vyberie jedna, ktorá bude nahradená dverami (obrázok b). Vybraná stena sa zmaže (obrázok c) a nahradí sa dverami (obrázok d).

4.4.4 Umiestnenie hráča

Poloha každej miestnosti je uložená. Pre každú miestnosť je uložený zoznam a poloha kociek, z ktorých sa skladá. Na základe toho je potom jednoduché, náhodne vybrať štartovaciu izbu, kde sa umiestni hráč a náhodne vybrať kocku, na ktorú bude postavený.

Na podobnom princípe sa vyberie aj koncová miestnosť. Tu ale miesto toho, aby sa hľadala podlaha na postavenie hráča, sa vyberie náhodná stena. Tá sa potom zmaže a nahradí dverami. Tieto dvere budú znamenať východ z bludiska. Aby dvere bolo možné otvoriť, musí hráč zozbierať všetky kľúče.

4.4.5 Umiestnenie objektov

Rôzne typy objektov sú rozdelené do niekoľkých skupín – kuchyňa, izba, kúpeľňa, obývačka. Tým sa dosiahne, že sú logicky rozmiestnené a nemôže sa stať, že sa záchod objaví v kuchyni.

V kuchyni sa môžu nachádzať predmety ako sú stôl, kuchynská linka, varič, v izbe zasa posteľ a v kúpeľni umývadlo, či sprchový kút.

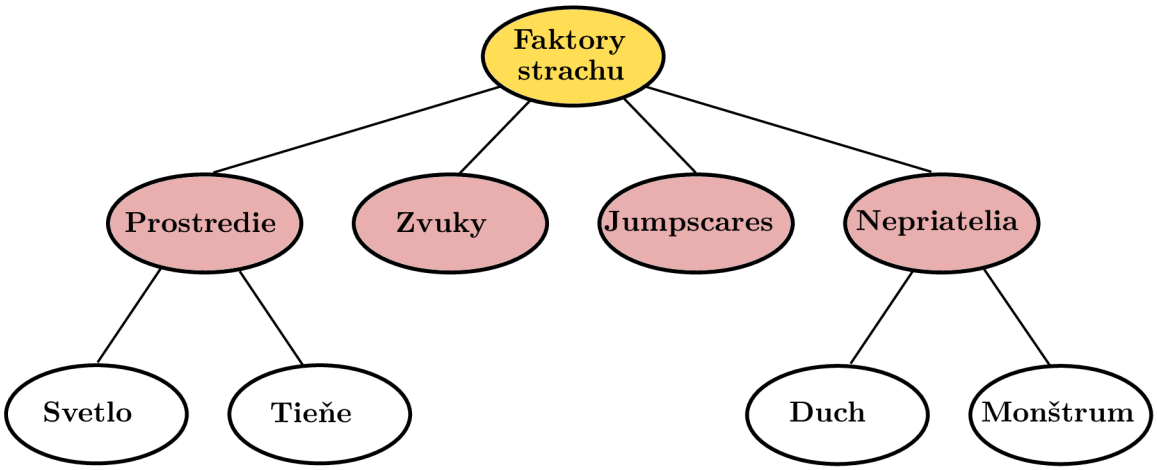
Zvolí sa izba zo zoznamu izieb, vyberie sa skupina objektov a predmety z tejto skupiny objektov sa rozmiestnia po izbe. Na základe kociek možno zistiť, ktoré časti izby sú obsadené a ktoré nie. Tak možno docieliť, že jednotlivé predmety sa neprekrývajú. Objekty zo skupiny sa nemusia rozmiestniť všetky. Ak je izba malá, môže sa tam umiestniť iba jeden objekt.

Podobne ako pri rozmiestňovaní izieb, aj tu možno jednotlivé kocky označiť ako obsadené alebo voľné. Vyberie sa kocka a na základe veľkosti predmetu sa rozhodne, koľko kociek okolo nej zaberie. Ak je priestor voľný, predmet sa umiestni a kocky sa označia ako obsadené.

Okrem predmetov, ktoré dotvárajú vzhľad jednotlivých izieb, dochádza ešte k rozmiestňovaniu kľúčov, baterky a predmetov, ktoré hráč môže zbierať do inventára (tužkové batérie, lekárnička). Tie sú náhodne umiestnené na náhodnú kocku v náhodnej izbe.

4.5 Návrh faktorov strachu

Najdôležitejšou súčasťou každej hororovej hry sú faktory strachu. Strach sa snaží byť vyvolaný či už atmosférou (ticho, tmavé prostredie, občasné náhodné zvuky), alebo aj okolitými elementami (obrázok 4.10).

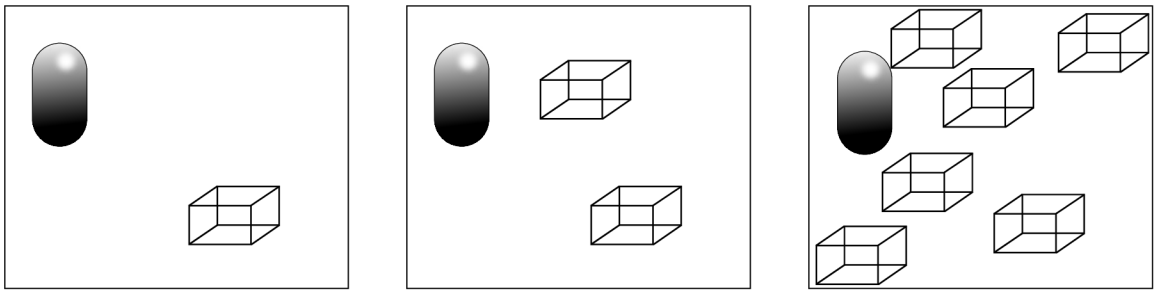


Obrázek 4.10: V hre sa nachádza niekoľko elementov, ktoré sa snažia vystrašiť hráča. Patria medzi nich hrôzostrašní nepriatelia, jumpscary, rôzne zvuky a blikajúce svetlá.

4.5.1 Jumpscare

Cieľom jumpscaru je hráča nečakane vystrašiť. Aby sa tohto efektu docielilo, bolo navrhnuté náhodne rozmiestňovanie kocky, do ktorej keď hráč narazí, tak sa spustí animácia. Na začiatku sa na náhodnom mieste v priestore vygeneruje jedna kocka. Zvolí sa počiatočný čas, v ktorom generovanie začne. Následne s časovým odstupom dvadsať sekúnd budú inštancie tejto kocky pribúdať na rôznych miestach vo vybranej miestnosti, (obrázok 4.11). Čím sa hráč bude dlhšie zdržiavať v miestnosti, tým sa jeho šance na vyskočenie jumpscaru zvyšujú.

Po narazení hráča do kocky sa všetky vytvorené kocky vymažú zo scény a na hráča vyskočí desivá animácia (obrázok 4.12).

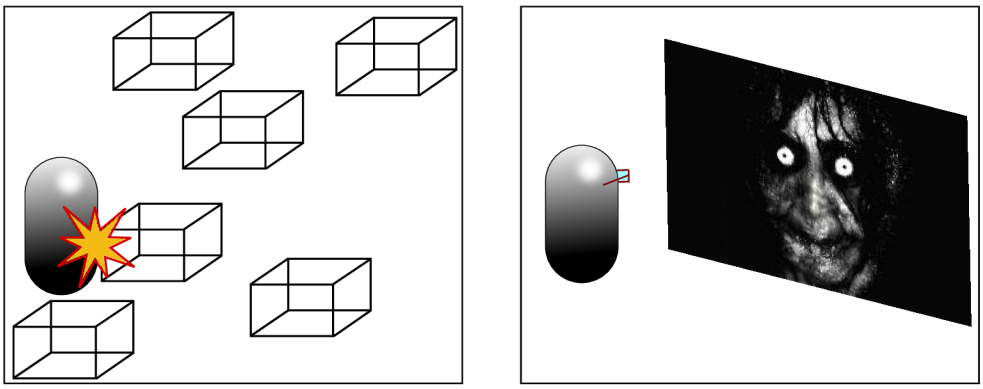


(a) Miestnosť v čase 20 sekúnd.

(b) Miestnosť v čase 40 sekúnd.

(c) Miestnosť v čase 120 sekúnd.

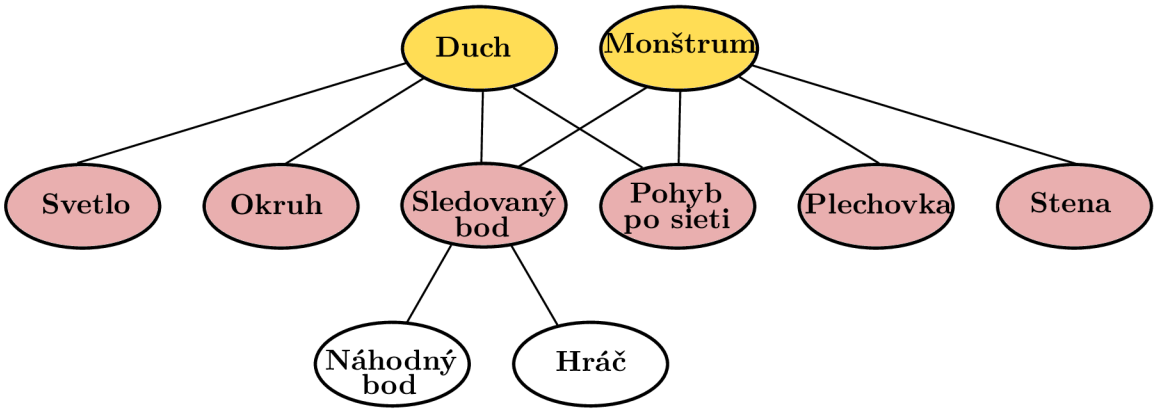
Obrázek 4.11: Každých 20 sekúnd pribudne v miestnosti nová kocka.



Obrázek 4.12: Po kolízií s kockou sa na obrazovke hráča zobrazí desivá animácia. Po tejto kolízií sa všetky kocky vymažú zo scény.

4.5.2 Útočiaci nepriatelia

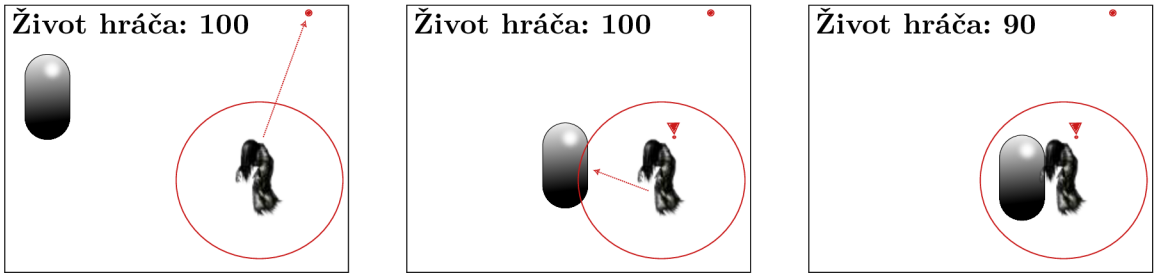
V hre sa hráč môže stretnúť s dvomi typy nepriateľov: duch a monštrum. Cieľom oboch týchto nepriateľov je zraniť hráča, v najhoršom prípade ho zabiť. Keď sa dostanú do určitej vzdialenosti od hráča, začnú ho zraňovať a jemu klesá úroveň života. Obaja nepriatelia majú definované plochy, po ktorých sa môžu pohybovať. Pohyb po tejto ploche je dosiahnutý vďaka bodu, ktorý prenasledujú. Poloha tohto bodu sa mení v čase a tak sa mení aj smer pohybu nepriateľov (obrázok 4.13).



Obrázek 4.13: V hre sa vyskytujú 2 nepriatelia. Každý z nich reaguje na iné podnety z okolia. Duch je citlivý na svetlo, zatiaľ čo monštrum vyprovokuje hocijaký zvuk, napríklad kopnutie do plechovky.

Prvým nepriateľom je duch. Duch môže chodiť cez zatvorené dvere a steny. Nemožno ho zabiť ani zneškodniť. Avšak je citlivý na svetlo.

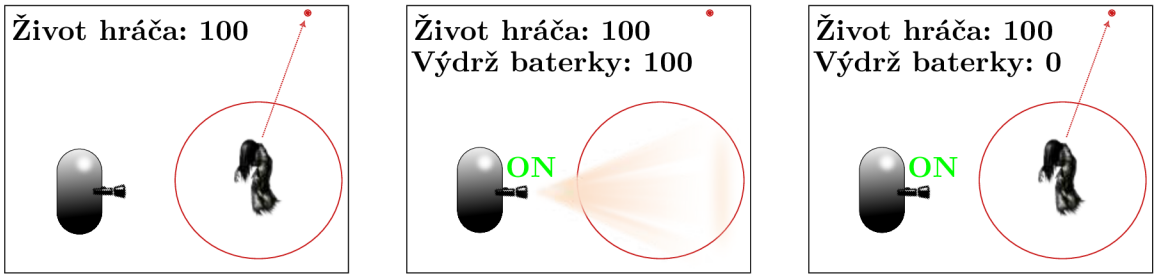
Pokiaľ vzdialenosť medzi ním a hráčom dosiahne určitú hodnotu, nový cieľ pre monštrum je hráč a začne ho prenasledovať namiesto kocky. Pri každom priblížení k hráčovi mu uberie život (obrázok 4.14).



(a) (b) (c)

Obrázek 4.14: Nepriateľ má svoj okruh, ktorý sníma, či v ňom hráč je alebo nie je. Pokiaľ v ňom nie je, prenasleduje bod. (obrázok a). Keď hráč vojde do duchovho priestoru, ducha prestane zaujímať bod a začne prenasledovať hráča (obrázok b). Keď je nepriateľ dostatočne blízko, začne na hráča útočiť a začne klesať úroveň života (obrázok c).

Keďže je nepriateľ citlivý na svetlo, hráč má možnosť v hracom priestore nájsť baterku. Keď baterka svieti, duch sa deaktivuje a hráč môže pokojne prejsť miestnosťou. Baterka má však svoju výdrž, ktorá sa v zapnutom stave každú sekundu zmenšuje. Keď dosiahne hodnotu 0, baterka prestane svietiť a duch sa znovu objaví. Hráč musí uniknúť monštru skôr, než ho zabije (obrázok 4.15).

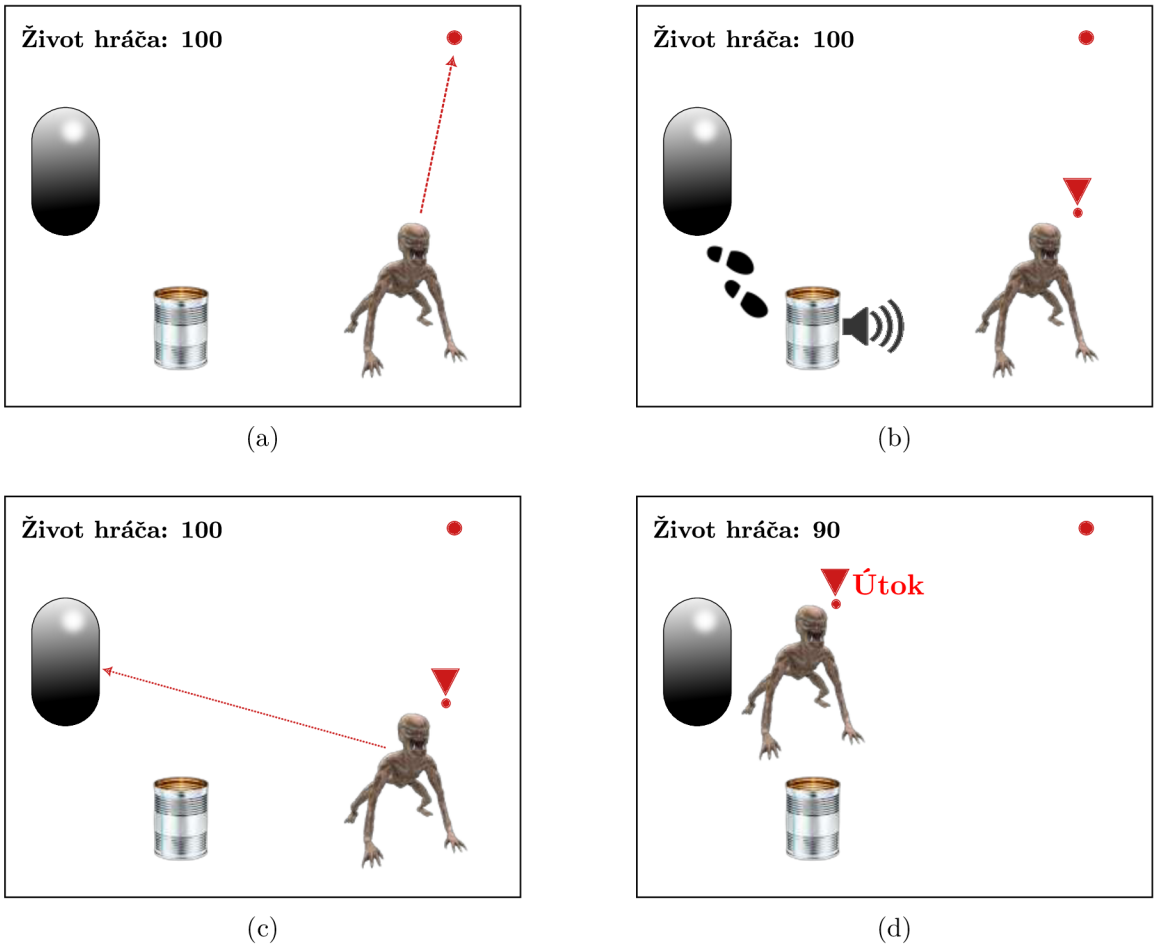


(a) (b) (c)

Obrázek 4.15: Hráč má možnosť použiť baterku. Keď nesvieti je možné vidieť ducha (obrázok a). Po zasvietení baterky duch zmizne a hráč môže bezpečne prejsť (obrázok b). Baterka má svoju výdrž, ktorá s časom klesá. Keď dosiahne hodnotu nula, baterka prestane svietiť a duch sa znovu objaví (obrázok c).

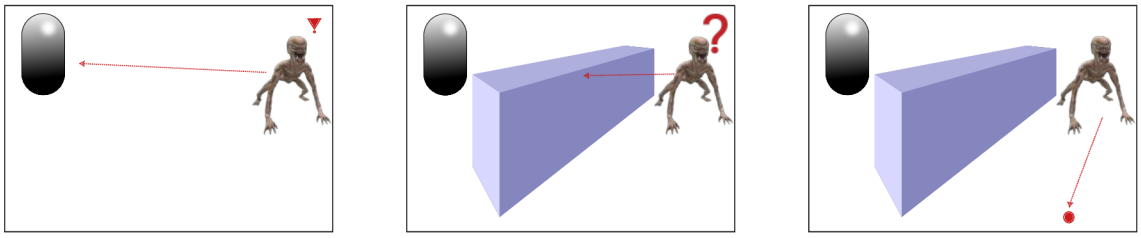
Druhým typom nepriateľa je monštrum, ktoré nevidí celkom dobre. Veľmi dobre však reaguje na hluk.

Podobne ako aj pri prvom nepriateľovi, aj tento má definovanú plochu, po ktorej sa môže pohybovať a prenasleduje bod. Za normálnych okolností sa pohybuje a žije si svojím životom. Ak hráč vydá zvuk (vrazí do plechovky) monštrum spozornie a začne hráča prenasledovať (obrázok 4.16). Hráč sa musí skryť za prekážku a potichu čakať, kým monštrum prenasledovanie vzdá.



Obrázek 4.16: Nepriateľ sa pohybuje voľne po ploche a prenasleduje bod, ktorého poloha sa v čase mení (obrázok a). Keď hráč vrazí do plechovky, vydá tak hlasný zvuk. Monštrum hneď spozornie (obrázok b). Nepriateľ sa miesto bodu zameria na hráča a začne ho prenasledovať (obrázok c). Keď je nepriateľ dostatočne blízko k hráčovi, začne ho zraňovať (obrázok d).

Medzi hráčom a monštrom existuje priamka. Pokiaľ do tejto priamky nevstúpi žiadny objekt, monštrum hráča stále vidí a prenasleduje ho. Pokiaľ priamka príde do styku s objektom, monštrum na chvíľu zastane a čaká. Ak sa hráč znovu neobjaví v zornom poli nepriateľa do určitého času, tak odchádza a hráč môže bezpečne prejsť ďalej (obrázok 4.17).

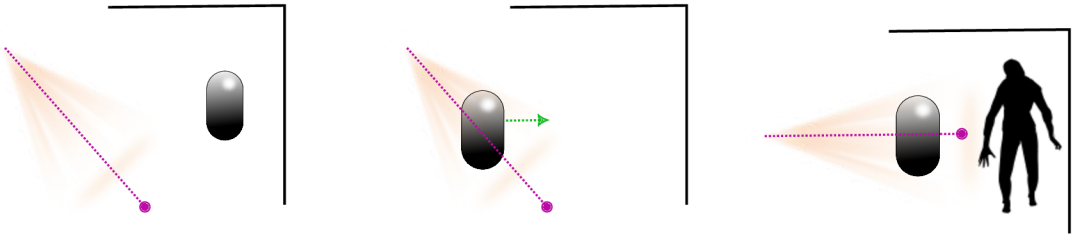


(a) (b) (c)

Obrázek 4.17: Medzi hráčom a nepriateľom existuje priamka, ktorá sníma či je medzi nim nejaký objekt alebo nie (obrázok a). Ak sa v ceste priamky objaví nejaký objekt, nepriateľ stratí výhľad na hráča a nevie ho viac zamerať (obrázok b). Monštrum začne opäť prenasledovať bod a hráča zanechá. (obrázok c).

4.5.3 Hra tieňov

Tiene sú dobrým nástrojom, ako zmiasť hráča a vyvolať v ňom dojem, že niekto za ním stojí. Pokiaľ je hráč v jednej rovine so svetlom a kamera smeruje tam, kam dopadajú lúče svetla, objaví sa na stene tieň (obrázok 4.18c). Pokiaľ ale trochu pohne kamerou a vyjde z tejto línie, neuvidí na stene nič (obrázky 4.18a a 4.18b).



(a) (b) (c)

Obrázek 4.18: Keď hráč nie je v jednej rovine so svetlom - nevidí nič (obrázok a). Keď je v jednej rovine so svetlom ale pozerá iným smerom - stále nevidí nič (obrázok b). Avšak keď hráč a jeho kamera sú v jednej rovine so svetlom - na stene sa zobrazí tieň (obrázok c).

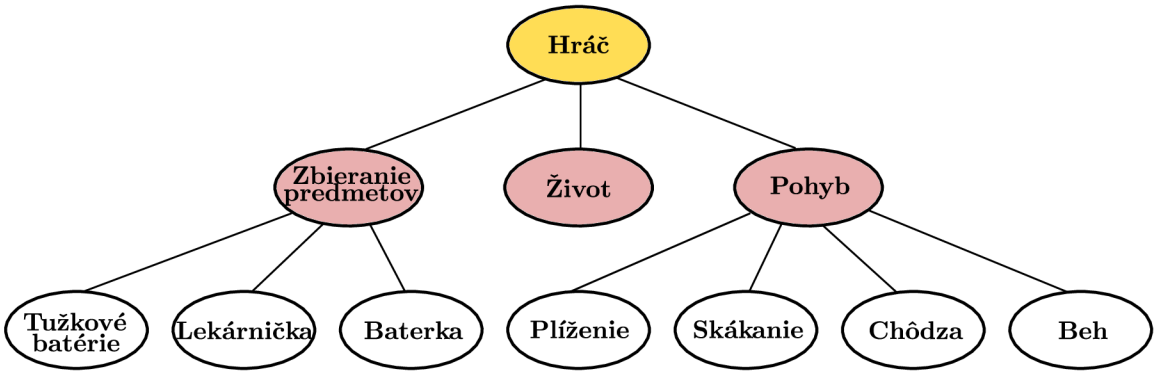
4.5.4 Prostredie

Jedným z dôležitým prostriedkov pri vytváraní hororovej atmosféry sú zvuky. Rôzne typy zvukov, ktoré majú v hráčovi vyvolať dojem, že sa niečo v jeho okolí nachádza sa rozmiestia po hracej ploche. Slúžia predovšetkým k zmäteniu hráča a vyvolaniu väčšieho napätia.

Prostredie tiež možno dotvoriť blikajúcim svetlom, kde sa intenzita svetla v čase náhodne posúva hore dole. Týmto spôsobom sa vytvorí blikajúci efekt.

4.6 Hráč

Hráč má svoj život, zbiera kľúče a predmety, otvára dvere a pohybuje sa po hracom priestore (obrázok 4.19). Môže chodiť, behať, skákať a zakrádať sa.



Obrázek 4.19: Hlavnou úlohou hráča v hre je chodiť po labyrinte a zbierať predmety. Aby hra bola ťažšia, má aj úroveň života, ktorý keď klesne na 0, tak hráč umiera.

Hlavnou úlohou hráča je hľadanie kľúčov, ktoré sú rozmiestnené po celom labyrinte. Aby bolo ovládanie jednoduchšie, v strede obrazovky sa nachádza kurzor. Keď s ním hráč zameria na konkrétny objekt, začne pulzovať a zobrazí sa informačná hláška, ktorá ho vyzýva k zdvihnutiu predmetu pomocou príslušného tlačidla prípadne k otvoreniu dverí.

Hráč môže zbierať lekárničky, ktoré mu dodajú život, baterky, tužkové batérie, kľúče, ktoré mu znovu umožnia použiť ich v hre.

Baterky a lekárničky zbiera do inventára, odkiaľ ich môže v prípade potreby použiť.

4.6.1 Zbieranie kľúčov

Cieľom hry je zozbierať všetky kľúče a ujsť z labyrintu preč. Aby mal hráč lepší prehľad, zobrazuje sa na obrazovke informácia o počte nazbieraných kľúčov. Kľúče, keďže ich nemožno priamo použiť, sa nezobrazujú v inventári.

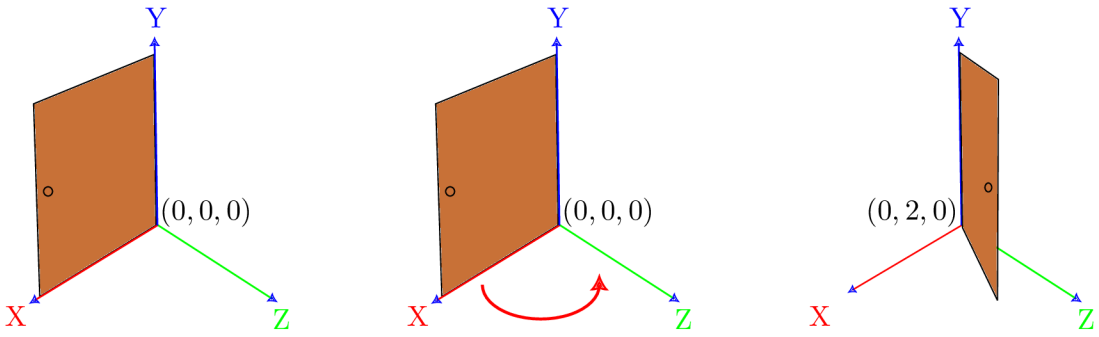
Kľúče majú svoje počítadlo. Po zdvihnutí kľúča sa počítadlo zvýši a objekt sa odstráni zo scény. Toto počítadlo sa hráčovi zobrazuje v ľavom hornom rohu. Získa tak jednoduchý prehľad o tom, koľko kľúčov má a koľko ich musí ešte nájsť (obrázok 4.20).

(a) (b) (c)

Obrázek 4.20: Hráč na obrazovke vidí informáciu o počte kľúčov. Kľúče sú rozhádzané náhodne po labyrinte (obrázok a). Keď hráč narazí na kľúč a zameria naň kurzorom, zobrazí sa mu na obrazovke výzva na stlačenie klávesy E (obrázok b). Po stlačení "E"objekt z obrazovky zmizne a na počítadle kľúčov sa stav zvýši. (obrázok c).

4.6.2 Otváranie dverí

Hráč môže rôzne interagovať s prostredím. Jedným z druhov interakcie je otváranie dverí. Existujú dva stavy, v ktorých sa môžu nachádzať – otvorené a zatvorené. Pri prechode z jedného stavu do druhého dvere rotujú okolo osi y (obrázok 4.21).



Obrázek 4.21: Princíp otvárania dverí. Dvere rotujú okolo osi y. Dvere sa otvárajú (rotujú) s rýchlosťou 2 jednotky okolo osi y.

4.6.3 Inventár

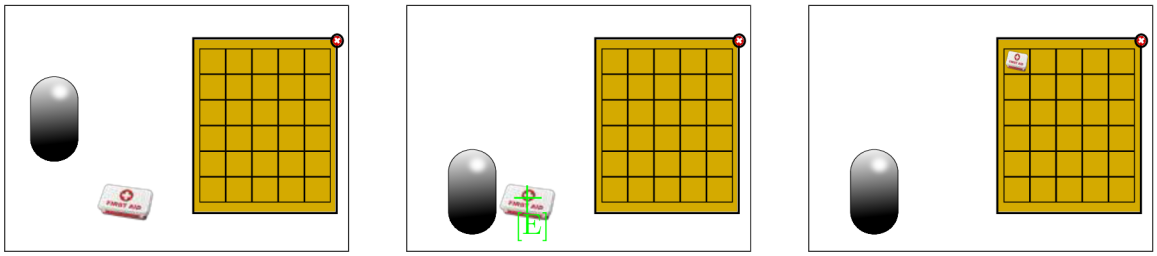
Ako už bol spomenuté, hráč má k dispozícií inventár, do ktorého si môže zbierať predmety (lekárnička, tužkové batérie). Inventár si môže hráč jednoducho zobraziť stlačením príslušnej klávesy. Keď je inventár otvorený, chod celej hry sa zastaví a hráč môže pracovať len s inventárom.

Inventár je vizualizovaný ako 2D objekt. Tento objekt má mriežku, do ktorej sú postupne pridávané ikony, symbolizujúce jednotlivé zozbierané predmety. Okrem iného inventár obsahuje tlačidlo, ktorým ho je možné zavrieť. Po zatvorení inventára sa chod hry znovu spustí (obrázok 4.22).

Každý objekt je jedinečne identifikovaný svojím ID, má svoj názov a hodnotu, ktorá predstavuje o koľko sa má zvýšiť hodnota života hráča alebo baterky.

Predmety sa po zodvihnutí zo scény vymažú a sú pridané do zoznamu objektov v inventári. Na základe názvu zdvihnutého predmetu sa vyberie ikona, ktorá sa zobrazí vo vizualizácií inventára spolu s príslušným názvom. Hráč kliknutím na túto novo vytvorenú ikonu je schopný objekt použiť. Po použití sa objekt vo vizualizácií a zoznamu objektov zmaže. Podľa typu objektu sa buď jeho hodnota pričíta k životu hráča alebo k životnosti baterky.

Objekt môže hráč priamo z inventára aj zmazať.

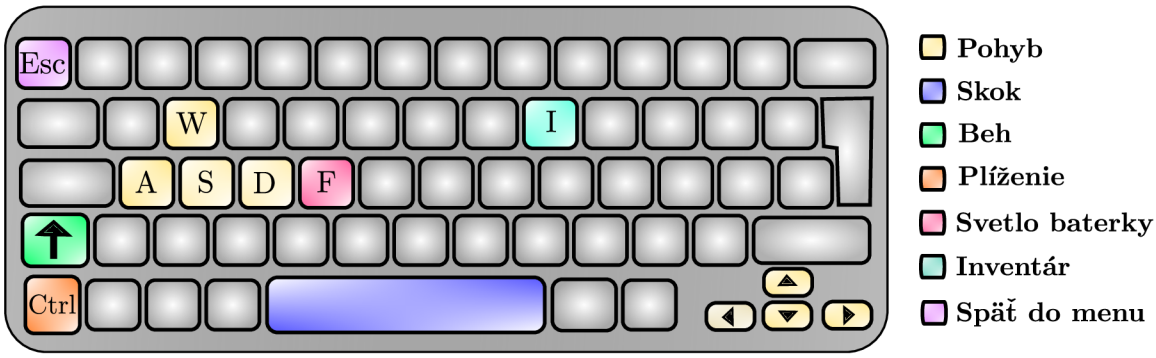


(a) (b) (c)

Obrázek 4.22: Hráč má k dispozícií inventár, do ktorého môže zbierať rôzne predmety – napríklad lekárničku (obrázok a) Keď zameria kurzor na zbieraný predmet, zobrazí sa mu na obrazovke výzva na stlačenie klávesy E (obrázok b). Po stlačení klávesy sa predmet objaví v inventári a zmaže sa z hry (obrázok c).

4.7 Ovládanie

Hra sa ovláda prostredníctvom myši a klávesnice (obrázok 4.23). Pohľad kamery sa hýbe spolu s pohybom myši. Samotný hráč sa pohybuje pomocou kláves W, A, S, D. Okrem klasického pohybu, môže hráč pri stlačení "Shift"bežať, "Ctrl"sa plaziť alebo "space"vyskočiť. Predmety je možné zbierať pomocou klávesy E. Baterku môže hráč zapnúť/ vypnúť pomocou klávesy F. Inventár sa zobrazí po stlačení klávesy I. Po otvorení inventára sa hráčovi na obrazovke zobrazí myš, pomocou ktorej môže buď použiť, alebo odstrániť predmety z inventára. Pokiaľ chce hráč hru ukončiť, môže tak urobiť pomocou tlačidla Esc. Tým sa vráti do menu.



Obrázek 4.23: Základné ovládanie hry pomocou klávesnice.

Kapitola 5

Unity

Pre jednoduchosť a veľkú popularitu bol pre implementáciu zvolený engine Unity. Jednotlivé scripty sú písané v jazyku C#. Kapitola vychádza z dokumentácie Unity [1]. Vybrané sú relevantné informácie, ktoré sa viažu k implementovanej hre. Unity je popísané tak, aby výsledná implementácia hry bola zrozumiteľná čitateľovi. Ďalšie informácie možno nájsť v publikáciach *Unity in Action: Multiplatform Game Development in C#* [5] a *Unity Game Development Cookbook: Essentials for Every Game* [10].

Unity je multiplatformový herný engine, ktorý je vyvinutý spoločnosťou Unity Technologies. Je považovaný za jeden z najpopulárnejších herných enginov na svete.

Jednou z najväčších výhod Unity je jeho používateľské rozhranie. Unity má veľmi intuitívne rozhranie a poskytuje vývojárom množstvo nástrojov a funkcií, ktoré im umožňujú vytvárať hry bez toho, aby museli mať veľa predchádzajúcich skúseností v programovaní alebo 3D grafike. Vývojári môžu používať prichystané zložky, tzv. assets a skripty, ktoré im pomáhajú pri tvorbe hry.

Vývojári môžu tiež využiť obrovskú komunitu Unity, ktorá sa delí o zdrojové kódy, prichystané zložky, tutoriály a iné užitočné materiály. Táto komunita umožňuje vývojárom zdieľať svoje skúsenosti, riešiť problémy a získavať nové znalosti a nápady.

Medzi jeho najdôležitejšie funkcie patria visual scripting, cross–platform development, 2D a 3D grafika, fyzika, umelá inteligencia, audio a networking.

Unity bol použitý pre vývoj niekoľkých populárnych hier, vrátane Hearthstone: Heroes of Warcraft, Ori and the Blind Forest, Monument Valley a mnohých ďalších.

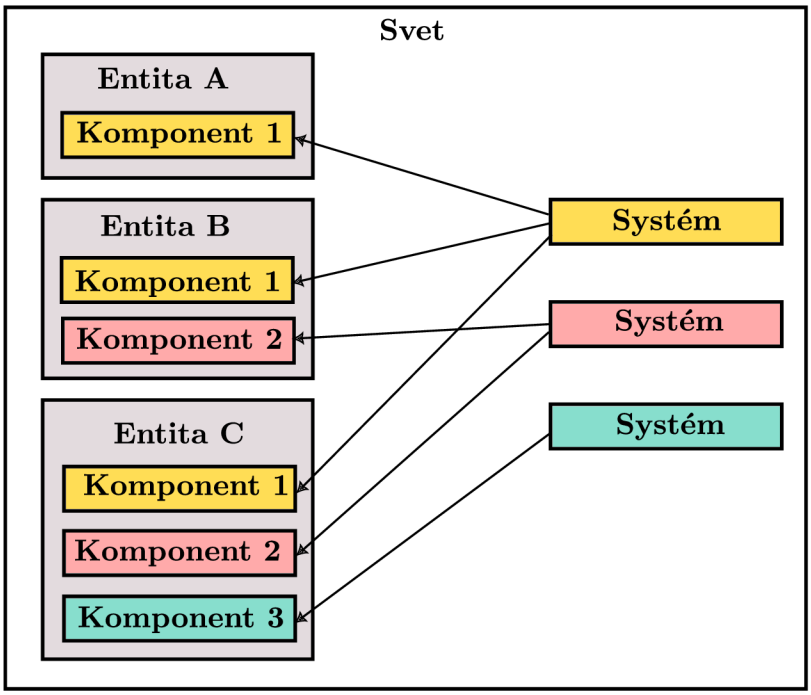
5.1 Entity component system

Jadrom Unity Data–Oriented Tech Stacku je entity component system. Unity Data-Oriented Tech Stack (DOTS) je skupina technológií a frameworkov, ktorých cieľom je optimalizovať a zefektívniť výkon hier a aplikácií postavených v Unity Engine.

Entity Component System (ECS) je programovací model, ktorý umožňuje vývojárom hier v Unity lepšie organizovať ich kód a dosiahnuť lepšiu výkonnosť. ECS nahradil tradičný model objektovo–orientovaného programovania pre hierarchické GameObjecty a komponenty v Unity.

Unity používa Entity Component System (ECS) na spravovanie dát v hrách. ECS umožňuje oddeliť dáta a logiku a spravovať ich oddelene, čo vedie k lepšiemu výkonu a efektívnosti v porovnaní s tradičným modelom programovania.

ECS v Unity sa skladá z entít, komponentov a systémov. Entita predstavuje konkrétny objekt v hre, zatiaľ čo komponenty sú dáta, ktoré popisujú entitu, napríklad polohu, rýchlosť, farbu a podobne. Systémy potom spracovávajú tieto dáta a vykonávajú akcie v hre, napríklad pohyb, kreslenie a iné (obrázok 5.1).



Obrázek 5.1: Model ESC, ktorý sa využíva pri vývoji hier v Unity. Jednotlivé entity reprezentujú herné objekty, ktoré sa vo svete nachádzajú. Každá enita je zložená z komponentov. Systémy, majú na starosti spravovanie jednotlivých komponentov. Vďaka tomu je možné paralerné vykonávanie úloh pre rôzne entity a ich komponenty.

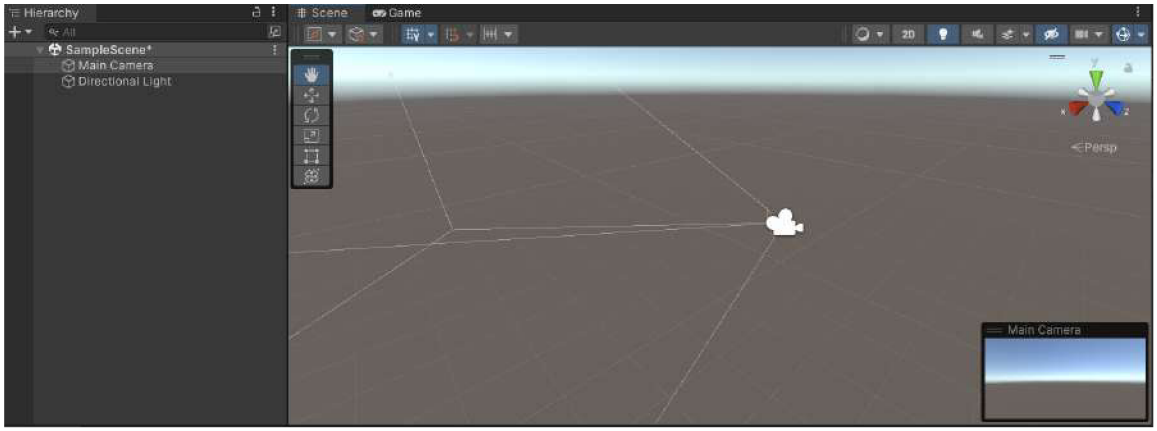
ECS umožňuje vývojárom vytvárať efektívnejšie a škálovateľnejšie hry, pretože oddelenie dát a logiky znamená, že jednotlivé časti hry môžu byť spracovávané paralelne, čo zlepšuje výkon. ECS tiež umožňuje vývojárom využívať moderné technológie, ako sú viacjadrové procesory a grafické karty, lepšie optimalizovať výkon hry pre rôzne platformy, ako aj vytvárať hierarchické závislosti a zároveň uchovávať priamu kontrolu nad svojimi dátami.

5.2 Scéna

Scéna v Unity je základným stavebným blokom pre tvorbu herného prostredia (obrázok 5.2). Je to virtuálny priestor, v ktorom môžu byť umiestnené herné objekty, postavy a vizuálne prvky. Scéna môže obsahovať rôzne prvky, ako sú osvetlenie, zvuky, kamery, terén, postavy, vozidlá a rôzne iné objekty a vizuálne prvky. Každý objekt môže mať svoje vlastnosti a parametre, ktoré môžu byť nastavené v editore Unity. Okrem toho sa v scéne nachádzajú aj rôzne komponenty, ktoré umožňujú interakciu s objektami a poskytujú im určité funkcie a vlastnosti.

V Unity je možné vytvoriť scénu pomocou grafického rozhrania alebo kódu. Scéna môže byť ľubovoľnej veľkosti a môže obsahovať rôzne typy objektov, ktoré navzájom medzi sebou interagujú. Tieto objekty môžu byť importované z externých programov, ako sú Blender,

Maya alebo 3D Studio Max, alebo je ich možné vytvoriť priamo v Unity. Hra môže mať viac scén a každá môže obsahovať iné objekty.



Obrázek 5.2: Základný vzhľad hernej scény v Unity. Základným prvkom každej hernej scény je kamera a osvetlenie. Do scény je potom možné pridávať rôzne objekty, efekty, animácie a pod.

5.3 MonoBehaviour

MonoBehaviour je základná trieda, z ktorej je odvodený každý skript Unity. Pri použití jazyka C# musí byť každý script explicitne odvodený z *MonoBehavior* triedy, inak ho nebude možné pripojiť k herným objektom.

Táto trieda poskytuje rôzne metódy, ktoré umožňujú programátorom manipulovať s pozíciou, rotáciou, mierkou a inými vlastnosťami herného objektu, ako aj komunikovať s inými hernými objektmi a s užívateľom.

Obsahuje funkcie pre inicializáciu, aktualizáciu, manipuláciu s komponentami, spracovanie kolízií a iné, ktoré umožňujú programátorom interagovať s engine, spravovať životný cyklus hry a spravovať herné objekty.

Medzi základné funkcie, ktoré boli použité aj pri implementácií patrí:

- *Start()* – Funkcia, ktorá sa spustí pri inicializácii herného objektu a slúži na inicializáciu premenných alebo nastavenie počiatočného stavu objektu. Volá sa len raz pri spustení hry.
- *Update()* – Funkcia, ktorá sa spustí každý snímok (frame) hry a slúži na aktualizáciu stavu herného objektu.

Medzi základné verejné metódy patrí:

- *Invoke()* – Ide o funkciu, ktorá zavolá zadanú metódu o určenom časovom s oneskorení v sekundách.
- *InvokeRepeating()* – Funkcia zavolá zadanú metódu po určitom časovom oneskorení v sekundách a potom ju opakuje podľa nastaveného intervalu.
- *CancelInvoke()* – Metódia, ktorá zruší všetky *Invoke* volania.

- *StartCoroutine()* – Ide o funkciu, ktorá umožňuje spúšťať korutiny. Korutiny sú metódy, ktoré umožňujú asynchrónné a časovo riadene spracovanie kódu. Ich vykonávanie je možné prerušiť a neskôr obnoviť.

- *StopCoroutine()* – Metóda slúži na zastavenie vykonávania korutiny, ktorá je aktívna vrámci aktuálneho objektu alebo skriptu.

Verejné metódy sa používajú pre spúšťanie a ukončovanie rôznych procesov. V *MonoBehaviour* triede možno tiež nájsť metódy, ktoré sa používajú na prístup k rôznym informáciam a komponentom vrámci herného objektu a jeho okolia.

Medzi najpoužívanejšie patria:

- *Instantiate()* – Táto metóda slúži na vytváranie kópií herných objektov. Je to veľmi užitočné na vytváranie nových objektov v hre, napríklad na vytvorenie nepriateľov alebo predmetov.

- *Destroy()* – Táto metóda slúži na odstránenie herného objektu. Je to užitočné pri eliminovaní nepriateľov, zničení predmetov alebo odstraňovaní objektov z hry.

- *GetComponent()* – Táto metóda slúži na získanie referencie na komponenty herných objektov. Pomocou tejto metódy môžeme pristupovať k vlastnostiam a metódam daného komponentu.

- *Play()* – Táto metóda slúži na spustenie prehrávania animácie v hernom objekte. Je to užitočné pri animácii postáv v hre.

V *MonoBehavior* triede sú dôležité aj metódy zvané *Messages*, ktoré sú volané automaticky v určitých situáciách počas behu hry. Tieto metódy slúžia na spracovanie rôznych udalostí v hre. Niektoré z najčastejšie používaných správ v Unity sú:

- *Awake()* – Táto správa sa volá, keď je herný objekt inicializovaný, ale predtým, ako sa zobrazí na obrazovke. Táto metóda sa používa na inicializáciu premenných a nastavenie počiatočných hodnôt.

- *OnTriggerEnter()* – Táto správa sa volá, keď sa herný objekt dostane do kontaktu s iným herným objektom, ktorý má nastavený Collider a Trigger. Táto metóda sa používa na spracovanie kolízií medzi hernými objektmi.

- *OnTriggerExit()* – Táto správa sa volá, keď herný objekt opustí zónu kolízie s iným objektom.

- *OnCollisionEnter()* – Táto správa sa volá, keď sa herný objekt dostane do kontaktu s iným herným objektom. Táto metóda sa používa na spracovanie kolízií medzi hernými objektmi.

- *OnMouseOver()* – Táto mettóda sa volá, keď sa myš dostane do konaktu s herným objektom.

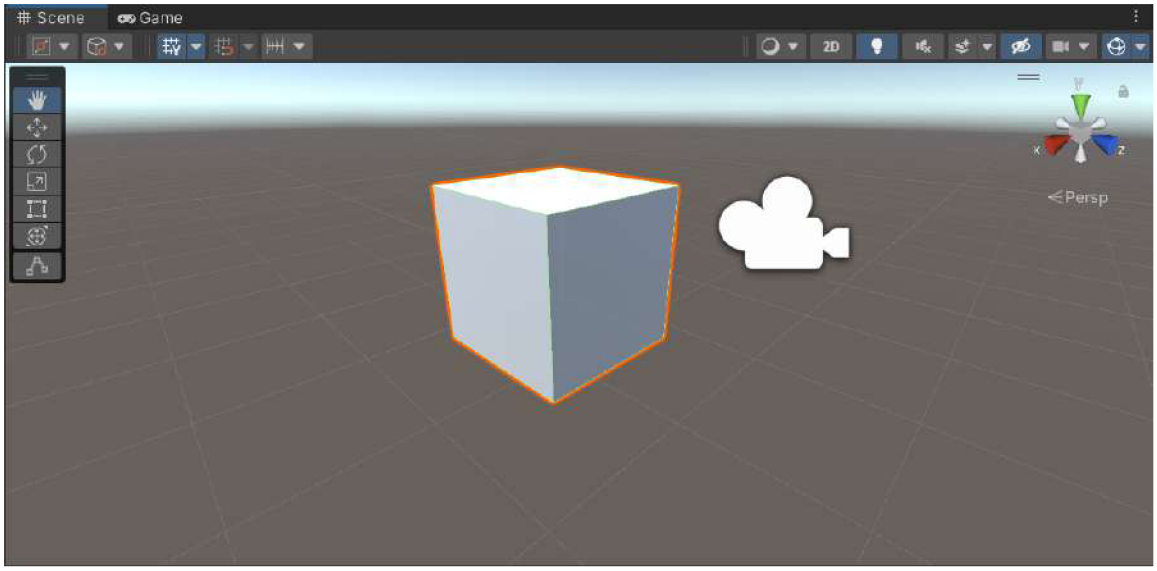
- *OnMouseExit()* – Táto metóda sa volá, keď sa myš už nie je viac v kontakte s herným objektom.

5.4 GameObject

Základným stavebným prvkom v Unity je *GameObject* (obrázok 5.3). Každý prvok v hre, ako napríklad postavy, predmety alebo prostredie, sú reprezentované ako *GameObjeckty*. Tieto objekty môžu byť pridané do scény a umiestnené v hierarchii objektov.

Každý *GameObject* má svoje meno, pozíciu, rotáciu a škálovanie v priestore 3D sveta, kde sa nachádza. *GameObject* môže byť umiestnený v hierarchii iného *GameObject*, ktorý mu môže slúžiť ako rodičovský objekt.

GameObject môže byť tiež prispôsobený pomocou skriptovania a grafického rozhrania Unity Editoru. Tieto prvky umožňujú tvorcom hier vytvárať vlastné funkcie a akcie pre *GameObject*, ktoré definujú jeho správanie a interakciu s okolitým prostredím.



Obrázek 5.3: Na obrázku je znázornená kocka, ktorá reprezentuje základný herný objekty v Unity.

5.4.1 Komponenty

Komponenty sú prvky, ktoré sa pridávajú k *GameObjectom* a definujú ich vlastnosti a správanie. Komponenty môžu byť grafické, zvukové alebo interaktívne prvky, ktoré umožňujú realizovať rôzne funkcionality v hre alebo aplikácii.

Každý komponent obsahuje vlastnosti, ktoré určujú jeho funkčnosť a interakciu s okolitým prostredím. Komponenty môžu byť pridané k *GameObjectu* pomocou Unity Editoru alebo skriptovania v jazyku C#.

GameObject v Unity obsahuje základné komponenty, ako napríklad *Transform*, *Renderer*, *Collider*, *Rigidbody*, *Mesh Renderer* a mnoho ďalších. *Transform* slúži na určenie pozície, rotácie a veľkosti objektu v 3D priestore. *Renderer* určuje, ako sa objekt zobrazuje v hernom svete, , *Mesh Renderer* zobrazuje vizuálnu reprezentáciu 3D objektu na základe jeho geometrie (meshu) a materiálov a *Collider* určuje jeho fyzikálne vlastnosti.

5.4.2 Collider

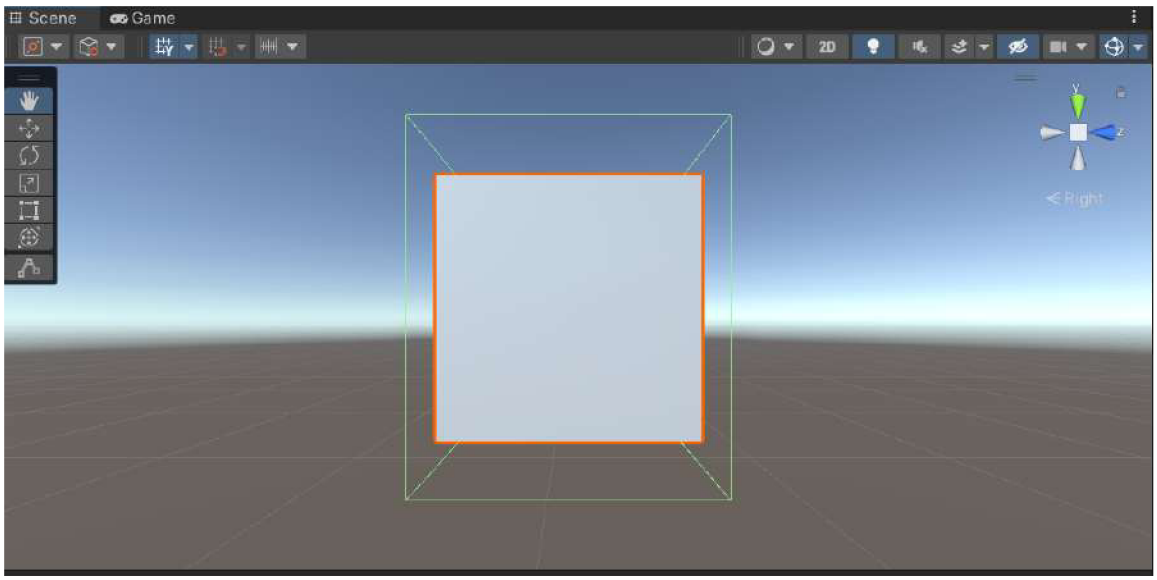
Collider (kolízny komponent) v Unity je komponent, ktorý sa pridáva k *GameObjectu* a slúži na detekciu kolízií s inými objektami v hre alebo aplikácii. *Collider* definuje tvar a

veľkosť kolíznej oblasti *GameObjectu*, ktorá je použitá na detekciu kolízií s inými objektami (obrázok 5.4).

Collider môže byť tiež použitý na detekciu kolízií s objektami z okolitého prostredia, ako sú terén, steny alebo iné objekty. Na základe toho možno určiť, ako sa bude herný objekt pri takejto kolízií správať. Napríklad, *Collider* môže byť použitý na zastavenie pohybu herného objektu pri kolízii s iným objektom alebo na spustenie skriptu, ktorý reaguje na kolíziu.

Existuje niekoľko typov kolíznych komponentov, medzi ktoré patrí:

- *BoxCollider*– Určuje kolíznu oblasť vo forme kvádra.
- *CapsuleCollider* – Určuje kolíznu oblasť vo forme kapsuly.
- *MeshCollider* – Určuje kolíznu oblasť na základe vlastností zvolenej 3D siete.



Obrázek 5.4: Herným objektom je možné pridávať rôzne kolízne komponenty. Na obrázku je zachytená kocka, ktorá obsahuje komponent *BoxCollider* (zelená kocka). Kolíznemu komponentu možno nastaviť veľkosť a umiestnenie vrámci objektu.

5.4.3 Rigidbody

Rigidbody v Unity je komponent, ktorý sa používa na definovanie fyzikálnych vlastností herného objektu, ako napríklad hmotnosť, gravitáciu a pohyb. Herný objekt môže tak reagovať na fyzikálne sily, ako sú gravitácia, sila alebo tlak. Použitie *Rigidbody* v hre alebo aplikácii výrazne zlepšuje realistickosť pohybu objektov a umožňuje simulovať realistickú fyziku.

Rigidbody má niekoľko vlastností, ktoré ovplyvňujú jeho správanie. Niektoré z týchto vlastností sú napríklad *Drag*, ktorý určuje odpor vzduchu pôsobiaci na herní objekt, *AngularDrag*, ktorý určuje odpor na rotáciu alebo *Mass*, ktoré určuje hmotnosť. Objekty s vyššou hmotnosťou reagujú na sily pomalšie ako objekty s nižšiu. Tiež sa budú pomalšie otáčať a pohybovať.

Rigidbody môže byť ovládaný pomocou skriptov, ktoré určujú jeho správanie pri rôznych interakciách s okolitým prostredím. Napríklad, skript môže byť použitý na pohyb herného objektu pomocou klávesnice alebo na spustenie animácie pri kolízii s iným objektom.

5.4.4 Navmesh

NavMesh v Unity je funkcia, ktorá umožňuje generovať navigačné siete pre objekty v hernom svete. Navigačná sieť sa tvorí z kolízií objektov, ktoré sa dajú prechádzať, a umožňuje objektom v hernom svete nájsť a plánovať cestu medzi dvoma bodmi. Oblasti sú vytvorené na základe geometrie scény a slúžia ako "cestné mapy"pre herné postavy, ktoré sa pohybujú po scéne.

V praxi sa *NavMesh* využíva v hre pre pohyb postáv po svete, kde umožňuje postavám pohybovať sa plynule a realisticky po rôznych terénoch a prekážkach. *NavMesh* poskytuje herným vývojárom rôzne možnosti nastavenia, ako napríklad rýchlosť postavy, vzdialenosť od prekážok a podobne.

Aby sa mohli herné objekty po navigačnej sieti pohybovať, musia mať priradenú komponentu *NavMesh Agent*. Tento komponent je schopný analyzovať *NavMesh* a nájsť optimálnu cestu medzi dvoma bodmi v hernom svete.

5.4.5 Scriptable object

Scriptable Object v Unity je trieda, ktorá umožňuje tvorcom hier a aplikácií vytvárať vlastné objekty so skriptovanými dátami, ktoré môžu byť následne použité v hre alebo aplikácii. *Scriptable Objecty* majú vlastnosti rovnaké ako ktorýkoľvek iný objekt v Unity, ale nevyžadujú, aby nie sú viazane k žiadnemu hernému objektu. To znamená, že umožňuje vytvárať a ukladať rôzne typy dát, ako napríklad inštancie tried, konfigurácie, textové súbory a ďalšie. Tvorcom to umožňuje vytvárať a spravovať rôzne dáta, ako sú zvuky, animácie, textúry, nastavenia a iné, bez nutnosti priraďovať ich k objektom v scéne.

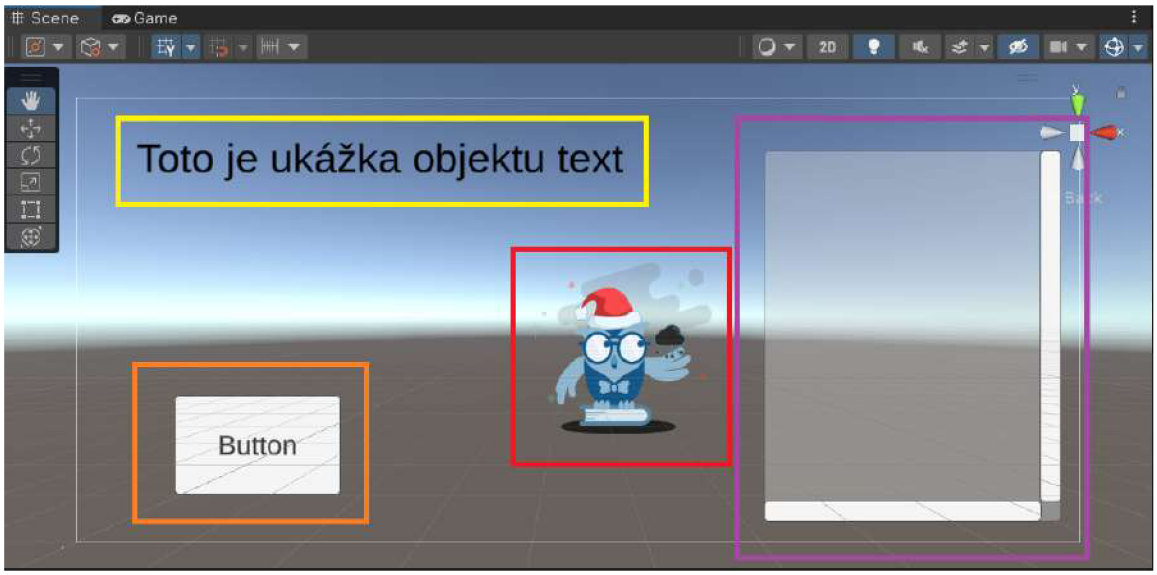
Scriptable Object umožňuje vytvoriť jednoducho zdieľané dáta medzi viacerými objektami v Unity hierarchii, ako aj medzi rôznymi scénami. Taktiež umožňuje vytvárať jednoducho editovateľné dáta, ktoré sa dajú uložiť a použiť v aplikácii.

5.5 UI komponenty

V Unity možno nájsť množstvo prvkov a nástrojov, pomocou ktorých je možné vytvoriť užívateľské rozhranie priamo v hre. Tieto UI komponenty umožňujú hráčom a užívateľom interagovať s hrou (obrázok 5.5).

Základné UI komponenty:

- *Canvas* – Je základným prvkom UI v Unity. Reprezentuje rovinu, na ktorú sú umiestnené všetky ostatné UI prvky. Obsahuje nastavenia ako je rozlíšenie, škálovanie atď.
- *Text(TextMeshPro)* – UI Text umožňuje zobrazovať statický text. Možno definovať jeho vlastnosti ako obsah, veľkosť, farba písma atď.
- *Image* – Slúži k zobrazeniu obrázkov alebo sprite.
- *Button* – UI *Button* je interaktivny prvok, na ktorý možno kliknúť. K tlačidlu možno priradiť udalosti (tzv. onClick), ktoré sú spustené pri kliknutí na tlačidlo.
- *Scroll View* – UI *Scroll View* umožňuje užívateľovi prehádzať obsah, ktorý je väčší než samotný *Canvas*. Podporuje horizontálne aj vertikálne posúvanie obsahu.
- *Layout Group* – *Layout Groups* sú komponenty, ktoré slúži k automatickému usporiadaniu prvkov vo vnútri panelu alebo iného kontajneru.



Obrázek 5.5: V unity prostredníctvom komponentu *Canvas* je možné jednoducho vytvoriť užívateľské rozhranie. Možno naň umiestňovať ďalšie komponenty. V žltom rámčeku je znázornený objekt *Text(TextMeshPro)*, v červenom rámčeku *Image* , v orandžovom rámčeku *Button* a vo fialovom rámčeku *Scroll View*.

5.6 Field of view

Field of View (FOV) je uhol pohľadu, ktorý určuje, aké objekty alebo oblasti sú viditeľné pre určitý objekt s kamerou. FOV sa používa najmä v 3D prostredí, kde zobrazuje, aké objekty sú videné v rovine, kde sa nachádza kamera.

FOV môže byť nastavený v kamere v Unity Editori alebo v skripte pomocou kódu. Pri nastavovaní FOV sa zvyčajne používa jednotka stupňov. Čím väčší je uhol FOV, tým širší je zorný uhol a tým viac sa bude zobrazovať v pohľade kamery. Naopak, menší uhol FOV znamená užší zorný uhol a zobrazí sa menej.

FOV sa používa najmä v rámci kamier a objektov, ktoré majú obmedzený výhľad, ako sú NPC postavy (Non-Player Character), príšery, bezpečnostné kamery a podobne. FOV je tiež dôležitý pri simulácii pohybu a viditeľnosti objektov v prostredí.

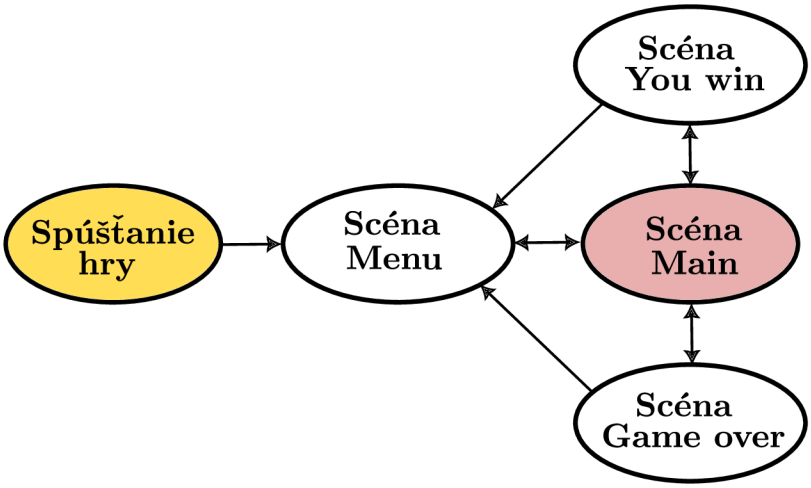
Kapitola 6

Výsledná implementácia

Táto časť popisuje výslednú implementáciu navrhnutých herných mechaník v Unity. Popisuje, ako je hra štruktúrovaná a rozložená do scén, z čoho sú jednotlivé scény zložené a ako sa postupne vytvárajú. Všetky textúry, modely a zvuky, použité vo výslednej implementácii, boli stiahnuté z verejne dostupných assetov z Unity asset storu [1].

6.1 Štruktúra aplikácie

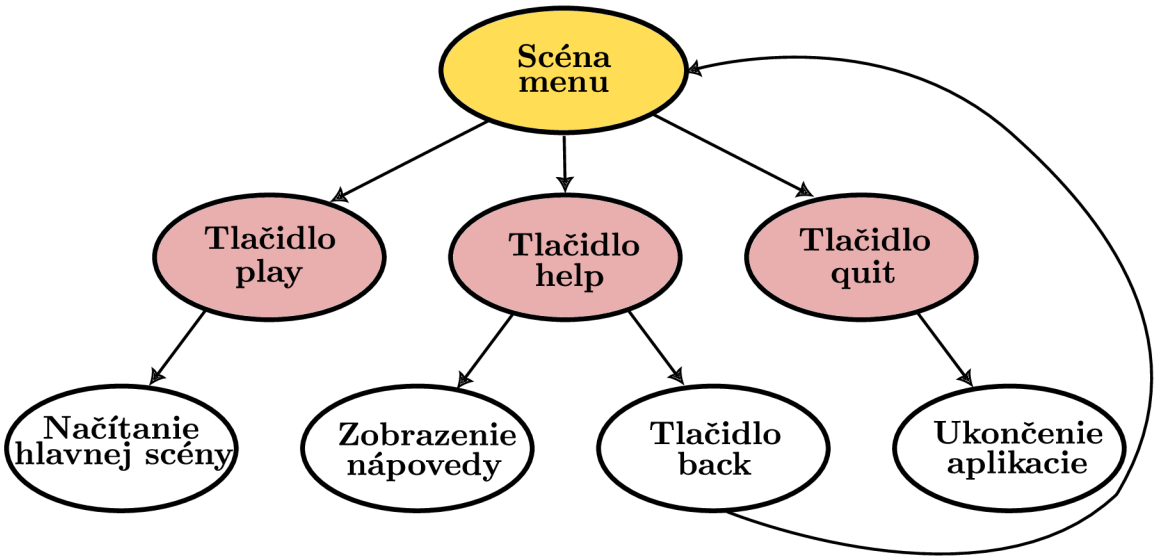
Hra obsahuje štyri scény: Scéna *Menu*, scéna *You win*, *Main* a scéna *Game over*, obrázok 6.1. Prechod medzi jednotlivými scénami je zabezpečený pomocou metódy *LoadScene*, ktorá slúži k načítaniu scény, ktorá jej bola zadaná v argumente. Všetky scény obsahujú komponent *Canvas*, osvetlenie a zvukový systém.



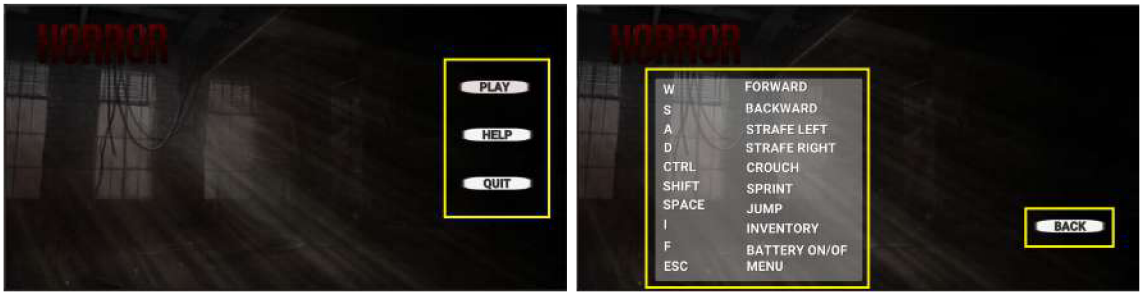
Obrázek 6.1: Všetky scény sú medzi sebou prepojené a možno medzi nimi prechádzať. Po zapnutí hry sa ako prvá zobrazí scéna *Menu*, z ktorej možno prejsť do scény *Main*. Podľa toho, či hráč hrou prešiel úspešne, alebo ho počas hrania zabili, sa zobrazí *You win* alebo *Game over* scéna. Z nich je možné sa opäť dostať do *Menu* alebo skúsiť hrať hru odznovu. Zo scény *Main* je možné hru aj ukončiť a dostať sa naspäť do scény *Menu*

[1] https://assetstore.unity.com

Menu, *Game over* a *You win* scéna sú jednoduché scény, ktoré zobrazujú len užívateľské rozhranie. Základnými komponentami týchto scén sú obrázky (*Images*), text (*TextMeshPro*) a tlačidlá (*Buttons*). Hlavným komponentom týchto scén sú práve tlačidlá, ktoré umožňujú hráčovi interagovať so scénami (obrázky 6.2 a 6.4) . Úvodnou scénou je scéna *Menu* (obrázok 6.3).

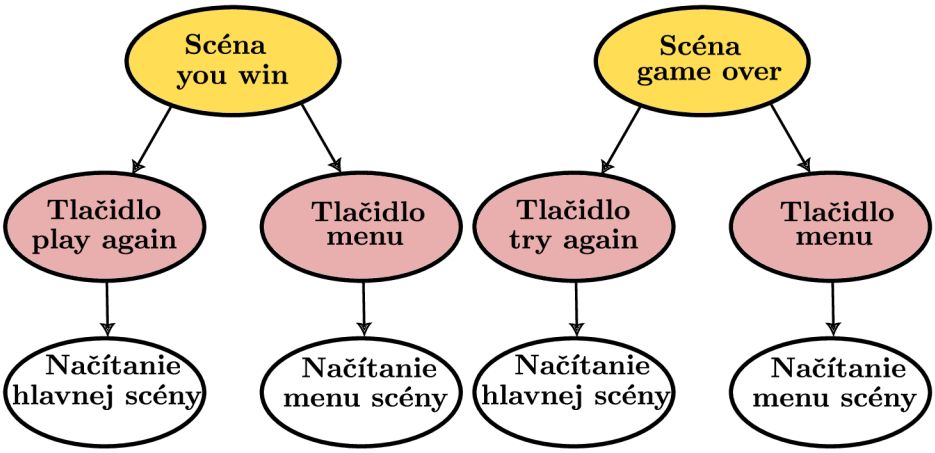


Obrázek 6.2: Scéna *Menu* obsahuje tri tlačidlá – *Play*, *Help* a *Quit*. Pomocou tlačidla *Play* sa načíta hlavná scéna a hráč tak začína hrať hru. Za pomoci tlačidla *Quit* sa ukončí celá aplikácia. Tlačidlo *Help* deaktivuje tlačidlá *Play*, *Help* a *Quit* a aktivuje tlačidlo *Back* a zobrazí nápovedu. Hráč tak nadobudne dojem, že sa prešlo do inej scény. Kliknutím na tlačidlo *Back* sa scéna vráti do úvodnej podoby.

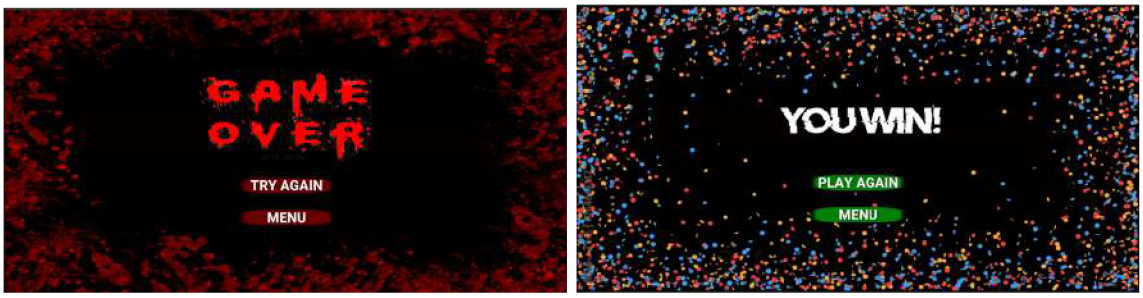


Obrázek 6.3: Výsledný vzhľad úvodného menu, vytvoreného v Unity. Obrázok v ľavo zobrazuje základný vzhľad. Obrázok vpravo zobrazuje užívateľské rozhranie, ktoré je zobrazené po kliknutí tlačidla help. Nápoveda je vyobrazená na komponente *ScrolledView*.

Scény *You win* a *Game over* majú totožnú funkcionalitu tlačidiel (obrázok 6.4). Líši sa len ich vzhľad (obrázok 6.5).

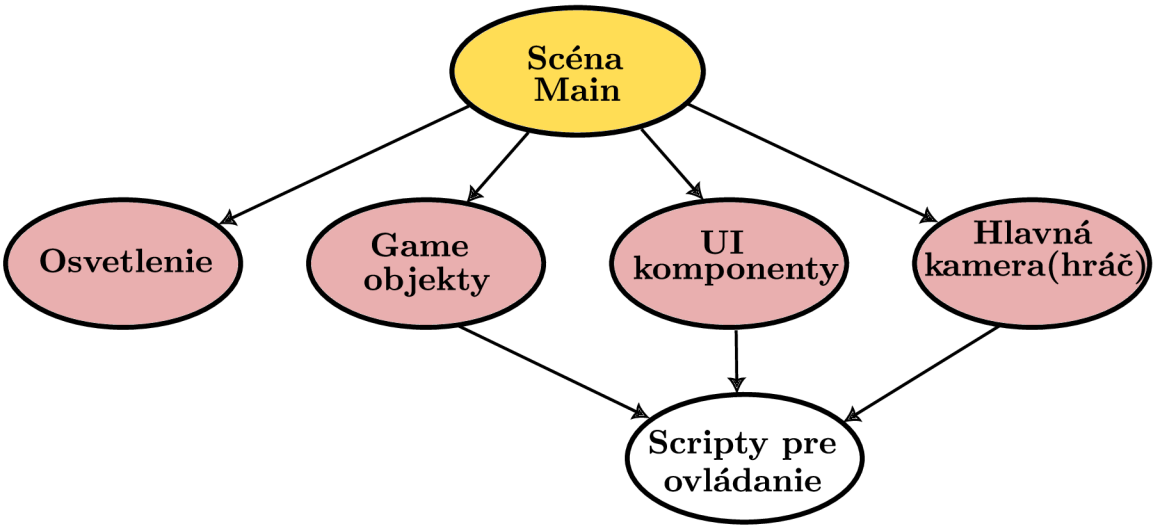


Obrázek 6.4: Scény *You win* a *Game over* obsahujú len dve tlačidlá – jedno na načítanie hlavnej hernej scény *Main* a druhé, ktorým je možné sa vrátiť do *Menu*.



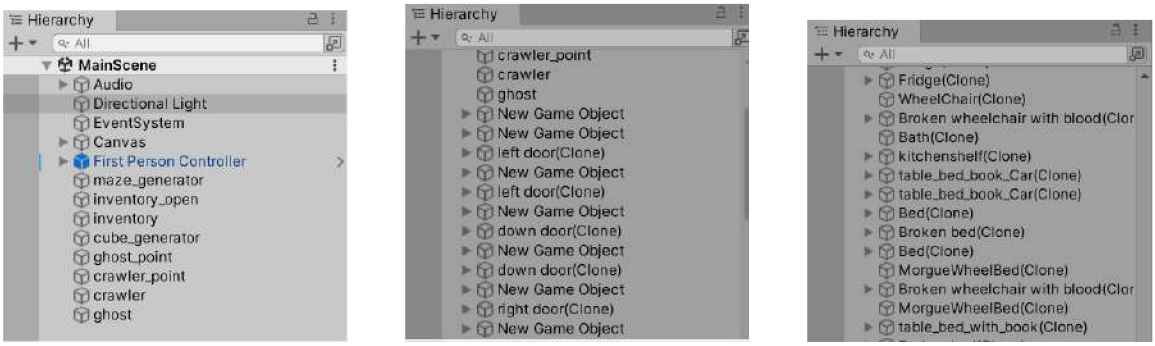
Obrázek 6.5: Výsledný vzhľad *You win* a *Game over* scény v Unity.

Hlavnou a najdôležitejšou scénou, je herná scéna *Main*, ktorá obsahuje celú logiku hry (obrázok 6.6). Nachádzajú sa v nej herné objekty, UI komponenty, osvetlenie a audio objekty.



Obrázek 6.6: Herná scéna *Main* je zložená z osvetlenia, ktoré sa snaží vytvoriť temnú atmosféru, game objektov, s ktorými hráč interaguje, UI rozhrania, pomocou ktorého má hráč vizualizovaný na obrazovke napríklad inventár a hlavnej kamery. Aby bola hra hrateľná, sú ku game objektom, UI komponentom a kamere priradené skripty, ktoré ich ovládajú a zabezpečujú chod hry.

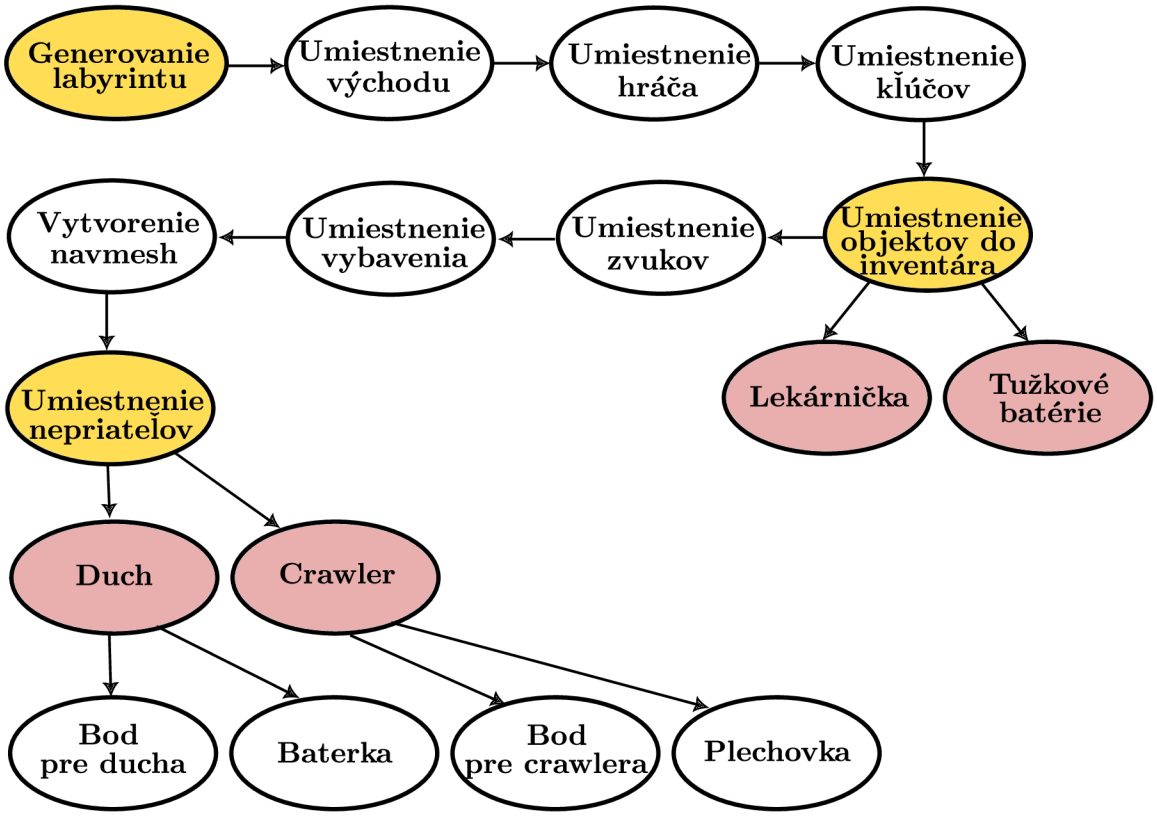
Keďže objekty sa generujú dynamicky (až po spustení aplikácie), herná scéna obsahuje len pár herných objektov a UI komponentov. Až po spustení aplikácie sa do scény umiestnia všetky potrebné objekty (obrázok 6.7).



(a) (b) (c)

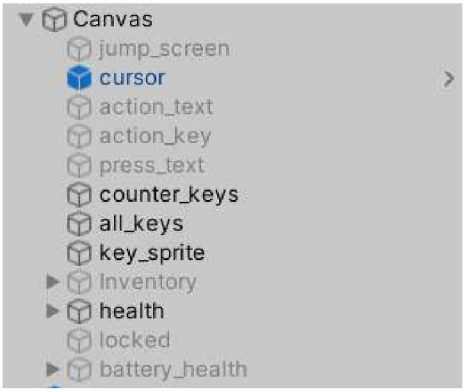
Obrázek 6.7: Scéna sa dynamicky vytvára po spustení hry. Pred jej spustením sa v nej takmer nič nenachádza (obrázok a) Po spustení hry sa do scény postupne pridávajú všetky objekty (b a c).

Najdôležitejším herným objektom hlavnej scény je *maze_generator*, ktorý má k sebe pripojený script, ktorý generuje celý labyrint a vyvoláva funkcie, ktoré postupne v rámci labyrintu rozmiestňuje jednotlivé herné objekty (obrázok 6.8).

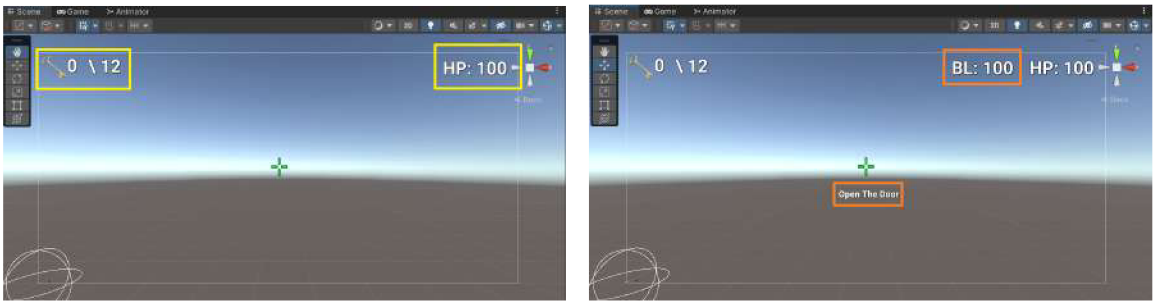


Obrázek 6.8: Do hernej scény sa postupne dynamicky umiestňujú jednotlivé objekty. Postupne sa v metóde *Start()* volajú funkcie, ktoré majú na starosti príslušný objekt umiestniť náhodne v labyrinte. Ako prvý sa vygeneruje celý labyrint. Keď je labyrint hotový, zavolá sa funkcia, ktorá vygeneruje v labyrinte únikové dvere. Po umiestnení dverí sa umiestní hráč, kľúče, objekty, ktoré hráč zbiera do inventára (lekárnička a tužkové batérie), zvuky a objekty, ktoré tvoria vzhľad izby (stoličky, stôl). Po umiestnení všetkých týchto objektov možno cez celý labyrint vytvoriť *NavMesh* a umiestniť naň nepriateľov. Spolu s nepriateľmi sa do scény umiestnia body, ktoré budú prenasledovať. S duchom sa do jednej miestnosti umiestní aj baterka a s crawlerom plechovka.

Ďalšími objektami sú *Audio*, ktoré obsahuje zvuky, ktoré hráč vydáva počas chôdze, behu a skoku, *Diretional Light*, *Canvas* (obrázky 6.9 a 6.10), *First Person Controller* a prázdne objekty *Crawler_point*, *Ghost_point*, ktoré sú zodpovedné za pohyb nepriateľov, *Ghost* a *Crawler*, ktoré vytvárajú pre nepriateľov navigačnú sieť, *Cubes_generator*, ktorý na sebe nesie script, ktorý je zodpovedný za generovanie kociek pre jumpscere a nakoniec *Inventory* a *Inventory_open*. Tie majú na sebe pripojené scripty, ktoré pomáhajú pri práci s inventárom.



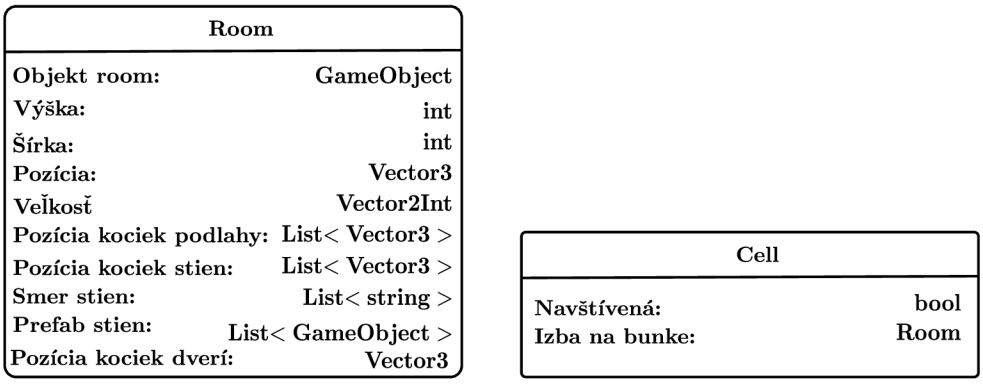
Obrázek 6.9: *Canvas* je zložený z niekoľkých objektov, ktoré sa hráčovi zobrazujú na obrazovke. Niektoré objekty sú zobrazené počas celého chodu hry, niektoré sa zobrazia na základe interakcií, ktoré hráč v hre vykonáva. Napríklad, keď hráč zameria kurzorom na dvere, zobrazí sa text *press_text*, ktorý vyzýva hráča k stlačeniu E.



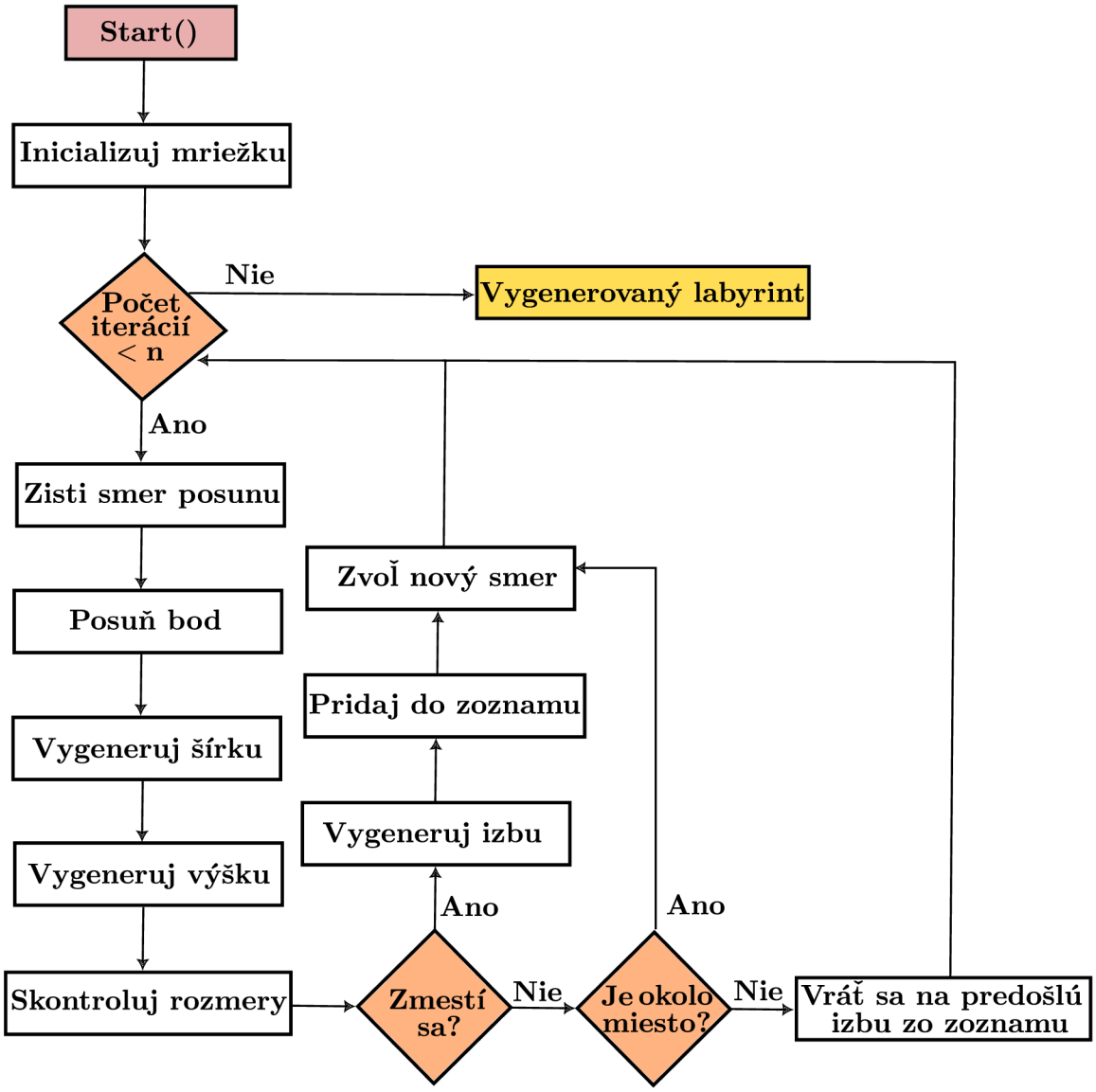
Obrázek 6.10: Výsledný vzhľad *Canvasu* v Unity. Hráč počas celej hry vidí na obrazovke kurzor, počítadlo kľúčov a svoj život. Postupne sa na obrazovke zobrazujú texty napríklad na otvorenie dverí alebo zostávajúca výdrž baterky.

6.2 Generovanie labyrintu

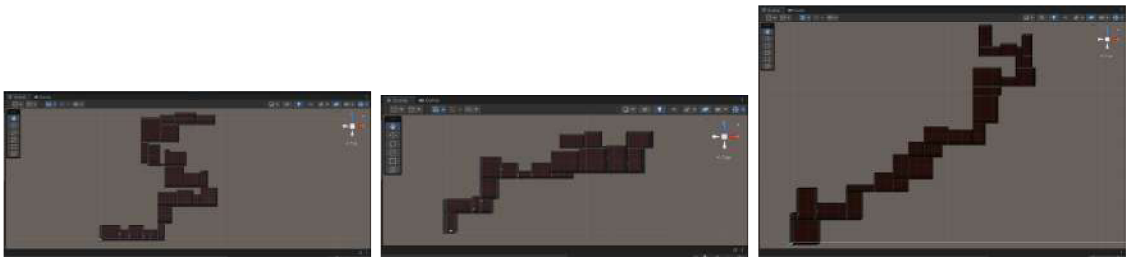
Najdôležitejšou a hlavnou časťou je generovanie labyrintu. Aby generovanie bolo jednoduchšie, boli vytvorené dve triedy – jedna pre uchovávanie informácií o stave jednotlivých buniek a druhá o stave izieb (obrázok 6.11). Hlavný algoritmus generovania je znázornený na obrázku 6.12.



Obrázek 6.11: Obrázok znázorňuje dve triedy, ktoré boli vytvorené pre jednoduchšiu prácu pri generovaní. Trieda *room* slúži na uchovávanie informácií o každej izbe. Medzi dôležité informácie patrí výška, šírka izby, jej veľkosť a pozícia, kocky a prefaby, z ktorých sa skladá. Trieda *cell* slúži na uchovávanie informácií o tom, či je bunka obsadená alebo voľná. Ak je obsadená, možno z nej zistiť, ktorá izba sa na nej nachádza,



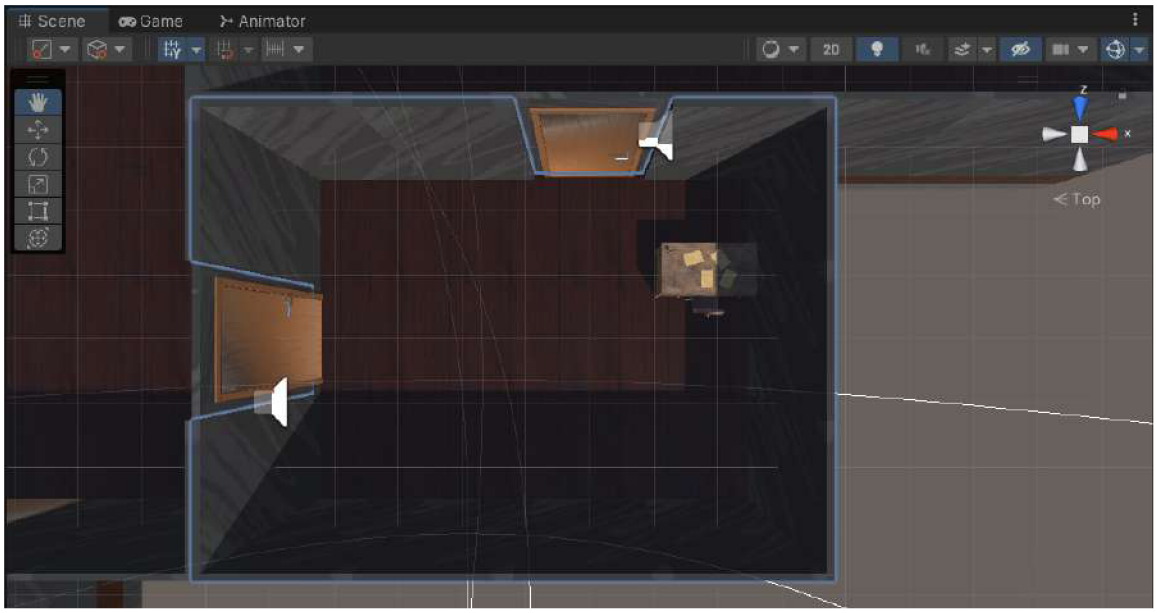
Obrázek 6.12: Obrázok zobrazujúci algoritmus generovania labyrintu. Všetko sa spúšťa v metóde *Start()*. Najprv dôjde k inicializácií mriežky, na ktorú sa budú ukladať jednotlivé vygenerované izby. Následne sa môže začať cyklicky prechádzať cez túto mriežku. Algoritmus končí, keď je mriežka celá vyplnená alebo v prípade, že počet iterácií nedosiahne hodnoty n. Podľa smeru, do ktorého sa od aktuálnej izby ide generovať izba, sa posunie začiatočný bod. Následne sa vygeneruje náhodná šírka a výška pre novú izbu. Tieto rozmery prejdú následnými kontrolami, aby sa zistilo, či sa izba na dané miesto, v danom smere môže umiestniť (nepresiahne grid?, nenachádza sa jej v ceste iná izba? atď). Ak sú rozmery vhodné a izba sa zmestí, môže dôjsť k vygenerovaniu izby. Ak sa nezmestí, prehľadá sa okolie v ďalších smeroch. Pokiaľ v okolí nie je žiadne miesto, nemá význam s izbou ďalej pracovať a je odstránená zo zoznamu izieb. Pokiaľ má izba okolo seba ešte miesto, vyberie sa nový smer pre generovanie.



Obrázek 6.13: Výsledný labyrint po generovaní v Unity. Po každom spustení hry sa labyrint vygeneruje inak.

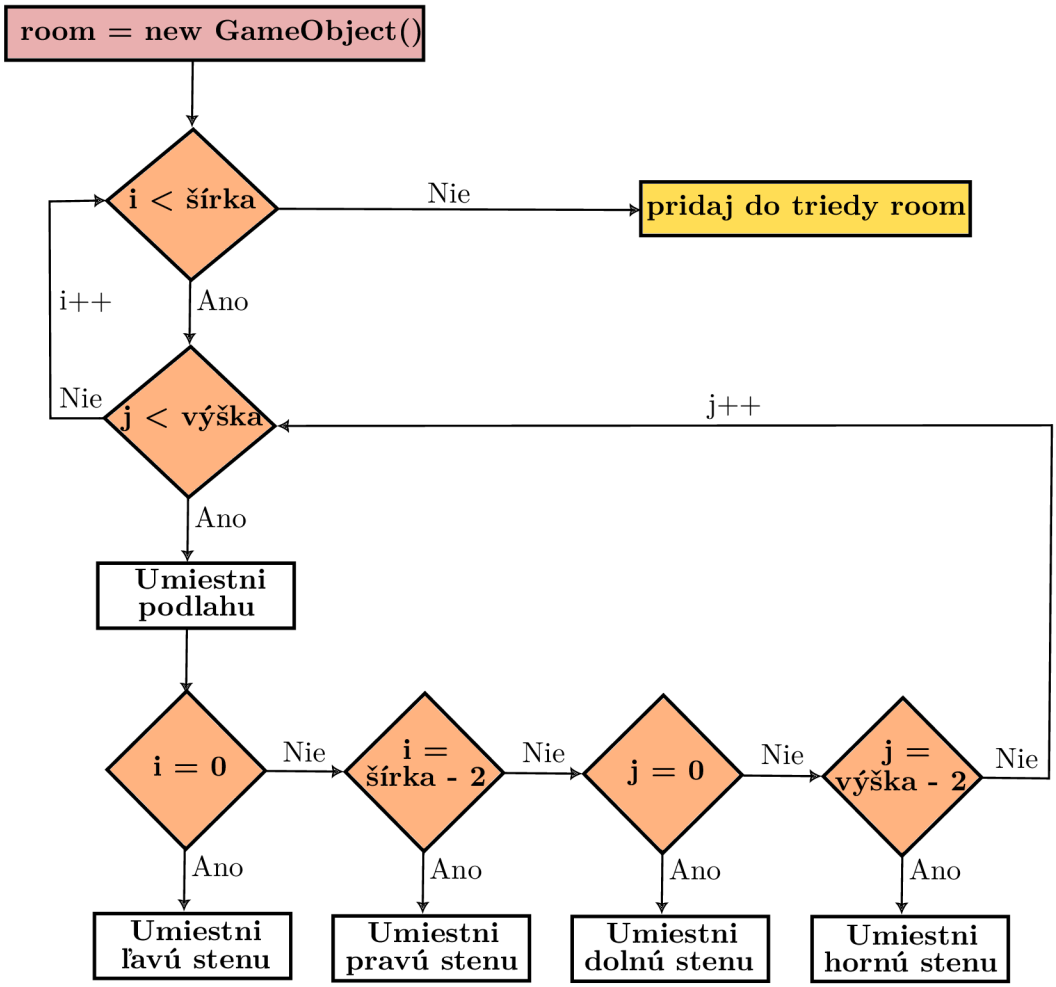
6.2.1 Generovanie miestnosti

Jednotlivé miestnosti sú zložené z kociek, ktoré sa postupne vedľa seba ukladajú (obrázok 6.14).



Obrázek 6.14: Výsledný vzhľad izby po jej vygenerovaní a umiestnení na správne miesto v Unity. Všetky izby sú štvorcového alebo obdĺžnikového tvaru.

Algoritmus vytvárania jednej izby je znázornený na obrázku 6.15. Kocky, z ktorých sa izby skladajú, boli vytvorené priamo v Unity editore (obrázok 6.16).

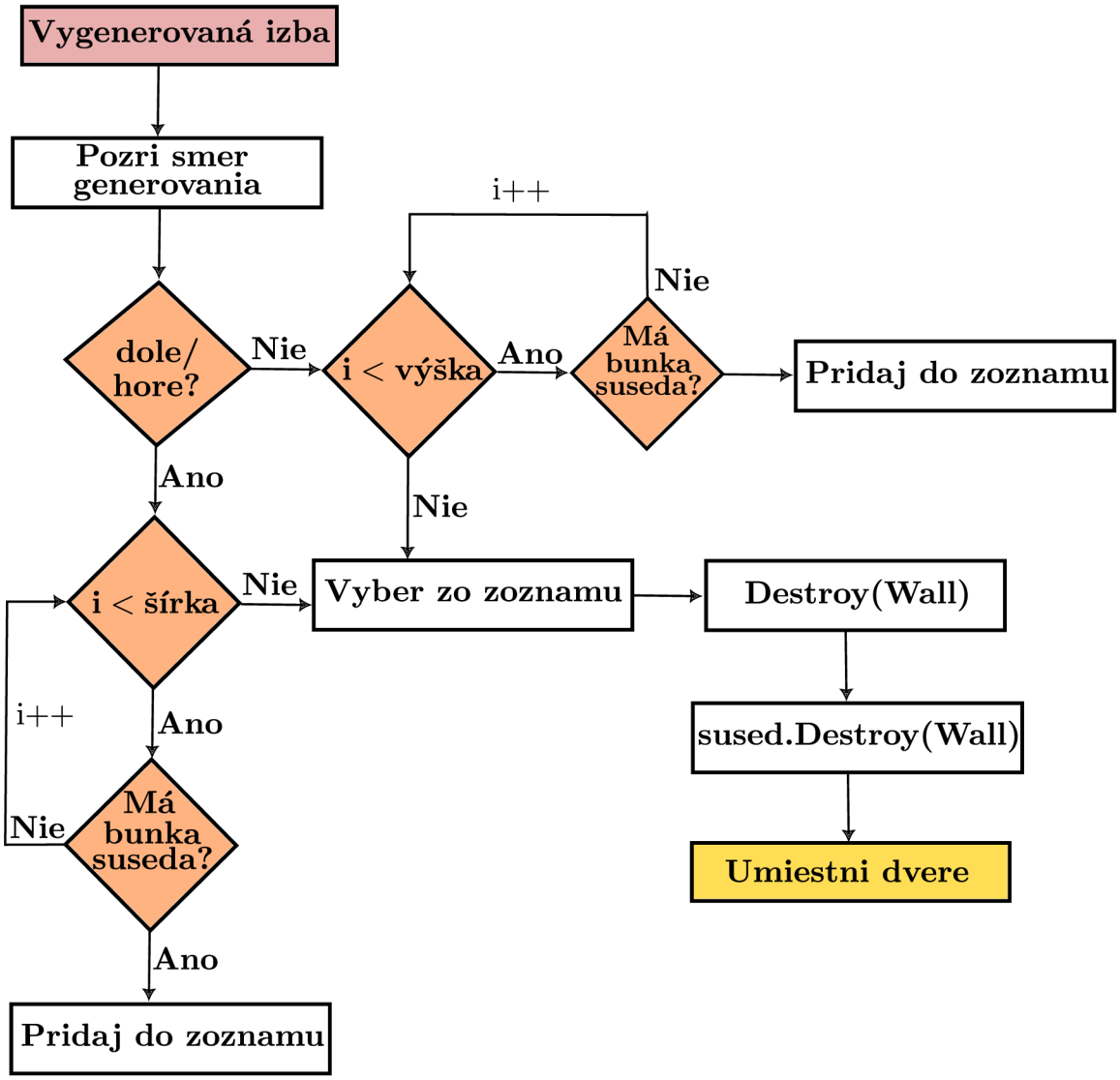


Obrázek 6.15: Obrázok zobrazuje algoritmus, ktorým sa vytvárajú jednotlivé miestnosti. Miestnosť nie je žiadnym statickým objektom, ale dynamicky sa pre každú novú miestnosť vytvára nový *GameObject room*. Po vytvorení tohto objektu sa iteratívne prechádza cez každý riadok a stĺpec miestnosti. Podľa aktuálnej polohy v mriežke sa kocka (prefab) umiestni na príslušné miesto. Keď je izba vytvorená pridá sa do zoznamu vytvorených izieb.

Obrázek 6.16: Izba je vytvorená z kociek, ktoré sa postupne vedľa seba umiestňujú. Existuje niekoľko typov kociek – podlaha, pravá/ľavá/horná/dolná stena a pravé/ľavé/horné/dolné dvere.

6.2.2 Umiestnenie dverí

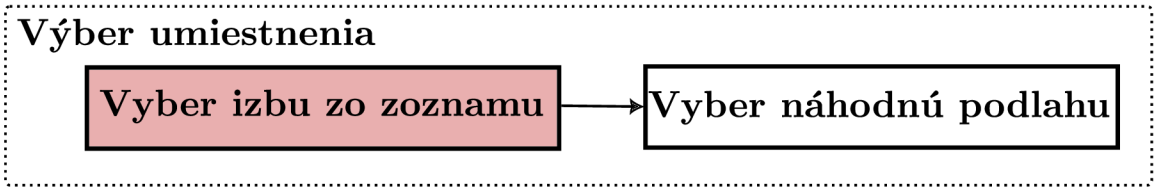
Po vygenerovaní miestnosti a jej následnom umiestnení, prichádza na rad umiestnenie dverí. Dvere sa vždy vytvárajú medzi novou izbou a izbou, vedľa ktorej sa generovala. Algoritmus umiestnenia dverí je znázornený na obrázku 6.17. Po tom čo sa vygenerujú dvere, môže algoritmus pokračovať v generovaní ďalšej miestnosti.



Obrázek 6.17: Obrázok popisuje algoritmus procesu umiestnenia dverí. Keď je izba vygenerovaná, nastáva proces umiestnenia dverí. Algoritmus sa najskôr pozrie, do ktorého smeru sa nová izba vytvorila. Podľa smeru prehľadá danú stranu miestnosti (ak sa generovalo do prava, prehľadá ľavú stenu). Ak sa generovalo nahor/nadol algoritmus prehádza po celej šírke steny a kontroluje, či každá bunka má suseda. Ak áno, pridá ju do zoznamu spoločných buniek. Pre smery naľavo/napravo sa prechádza po výške. Keď sa prejde celou stenou, vyberie sa zo zoznamu jedna náhodná bunka, na ktorej je umiestnená stena. Tá sa následne zmaže z izby a potom aj zo susednej izby. Prázdny priestor sa nahradí dverami.

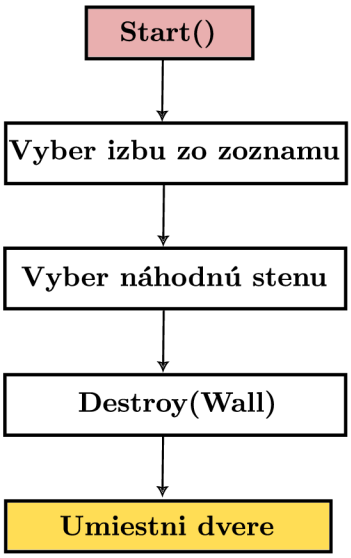
6.2.3 Umiestnenie herných objektov

Keď je celý labyrint vygenerovaný, prichádza na rad umiestnenie koncových dverí, hráča, nepriateľov a objektov. Základný algoritmus pre umiestnenie týchto objektov je rovnaký (algoritmus 6.18).

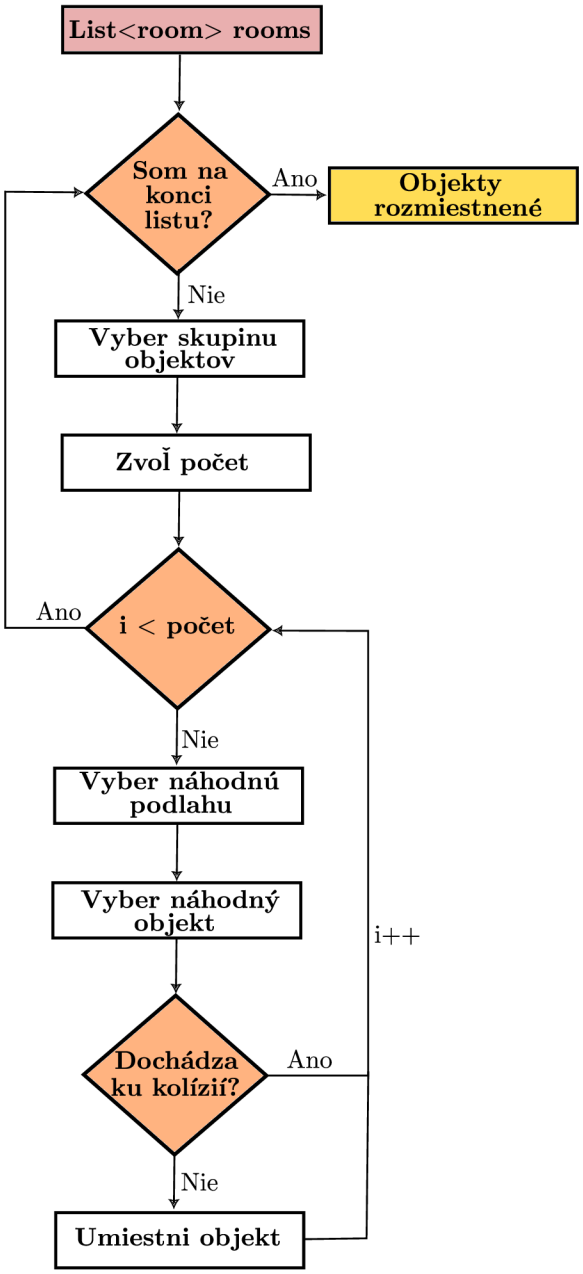


Obrázek 6.18: Základný algoritmus pre umiestnenie hráča, kľúčov, objektov, ktoré možno zbierať do inventára a nepriateľov je rovnaký. Na začiatku sa vyberie jedna náhodná miestnosť zo zoznamu izieb. Následne sa vyberie jedna náhodná bunka, ktorá reprezentuje podlahu.

Výnimkou je len algoritmus pre umiestnenie dverí, kedy dochádza k hľadaniu steny (algoritmus 6.19) a umiestnenie objektov (obrázok 6.21), ktoré len dotvárajú vzhľad miestností ako sú stoličky, stôl a podobne (algoritmus 6.20).

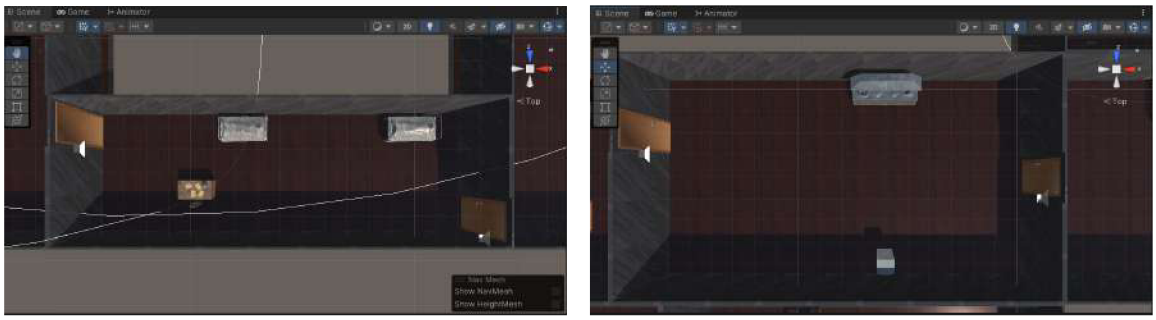


Obrázek 6.19: Algoritmus umiestnenia koncových dverí. Po vygenerovaní celého labyrintu sa umiestnia únikové dvere. Algoritmus zo zoznamu izieb vyberie náhodne jednu. Prejde jednotlivé steny vybranej izby a náhodne vyberie jednu stenu, ktorú zmaže. Na zmazané miesto následne umiestni dvere.

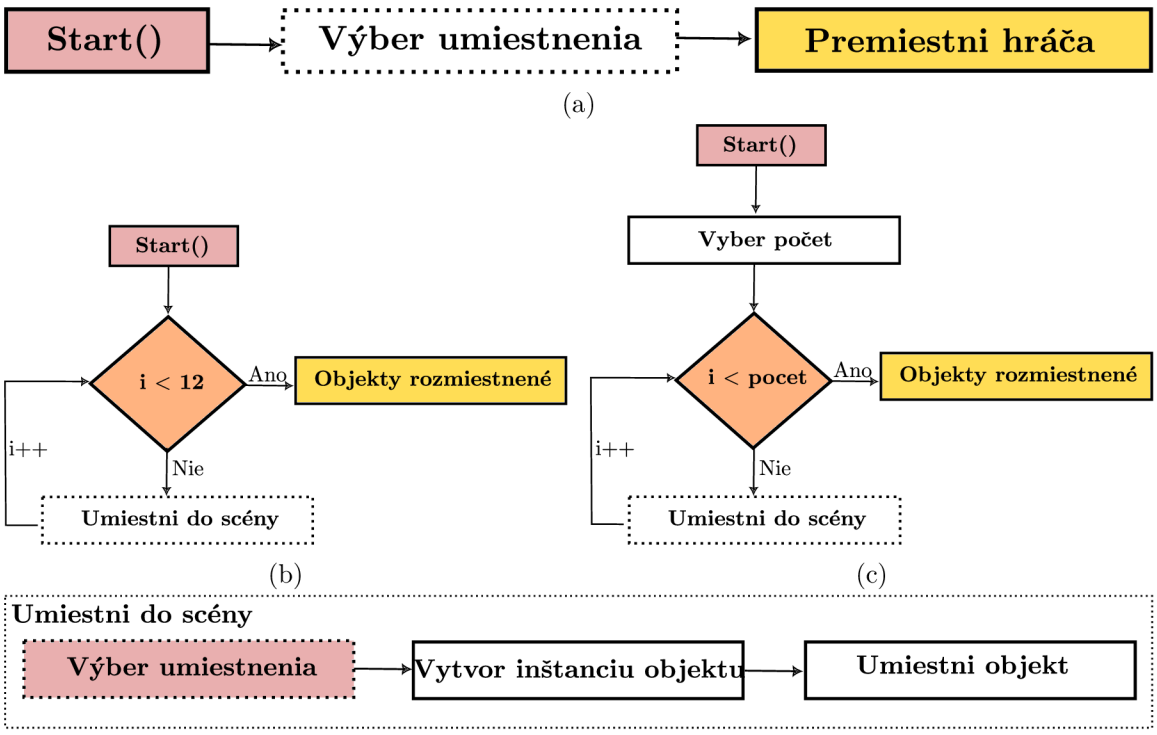


Obrázek 6.20: Obrázok znázorňuje algoritmus, ktorý rozmiestňuje jednotlivé objekty po labyrinte. Algoritmus, na rozdiel od ostatných, nevyberá náhodnú izbu ale postupne prechádza izbu po izbe a umiestňuje do nich objekty. Jednotlivé objekty sú rozdelené do 4 skupín – obývačka, kuchyňa, spálňa a kúpeľňa. Z tejto skupiny sa náhodne zvolí jedna. Následne sa zvolí náhodný počet objektov, ktoré sa budú do miestnosti generovať. Cyklicky sa prechádza týmto počtom a zakaždým sa zo skupiny vyberie náhodne jeden objekt, ktorý sa umiestní na náhodne vybranú podlahu v miestnosti. Pred umiestnením objektu dochádza najskôr ku kontrole, či v mieste umiestnenia nedochádza ku kolízií s iným objektom, prípadne stenou. Ak áno, objekt v tom prípade umiestnený nebude.

Umiestnenie hráča, kľúčov, lekárničiek a tužkových batérií je takmer totožné (obrázok 6.22).



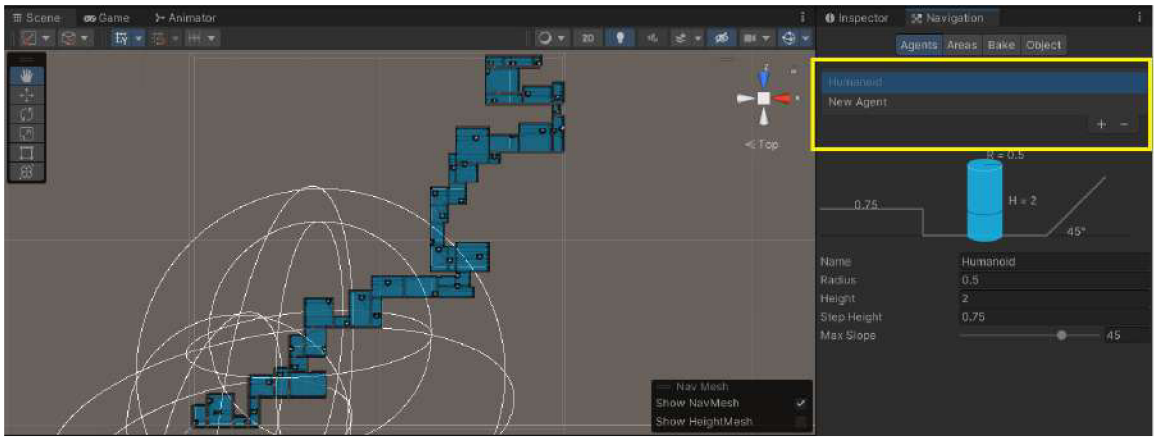
Obrázek 6.21: Výsledné rozmiestnenie objektov, ktoré dotvárajú vzhľad jednotlivých miestností



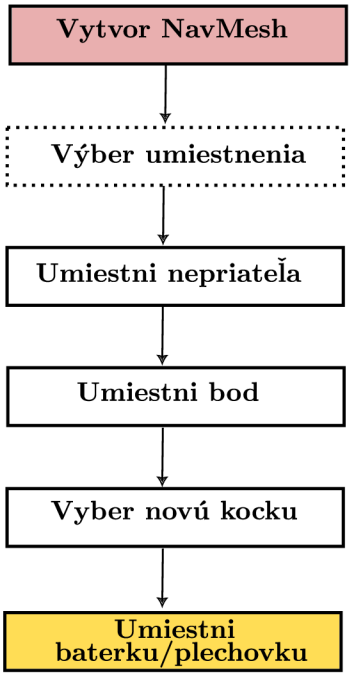
Obrázek 6.22: Algoritmi zobrazujú proces umiestnenia hráča, kľúčov a predmetov, ktoré hráč zbiera, po labyrinte. Keďže hráč je pred spustením v hernej scéne, stačí len zmeniť jeho polohu podľa toho, kde sa nachádza vybraná podlaha (obrázok a). Pri umiestňovaní lekárničiek, kľúčov a tužkových batérií dochádza cez metódu *Instantiate()*, k vytvoreniu kópií, ktoré sú následne náhodne rozložené po podlahe rôzne v labyrinte. V prípade kľúčov sa vytvorí 12 kópií (obrázok b). Počet lekárničiek a tužkových batérií sa volí náhodne (obrázok c). Objekty sa ale následne rovnakým spôsobom umiestnia do scény (obrázok d).

Pre nepriateľov je pred ich umiestneným potrebné vytvoriť navigačnú sieť, po ktorej sa môžu pohybovať. Táto sieť je vytvorená pomocou nástroja *NavMeshSurface*. Keďže každý nepriateľ sa pohybuje inak (jeden môže prechádzať cez steny a objekty, druhý nie), sú vytvorené dva separátne *NavMeshe* (obrázok 6.23). Objekty je možné rozdeliť do niekoľkých vrstiev. Objekty sú priradené do separátnej vstvy a tak isto aj steny. Jeden *NavMesh* pri

vytváraní vynecháva vrstvu, na ktorej sa nachádzajú objekty a steny. Tým je docielené, že *NavMesh* sa vytvorí aj cez tieto objekty a nepriateľ tak môže nimi prejsť. Keď je *NavMesh* vytvorený, môžu sa umiestniť nepriatelia (algoritmus 6.24).



Obrázek 6.23: Aby sa nepriatelia mohli pohybovať po ploche, je potrebné vytvoriť navigačnú sieť. Existujú dva typy sietí – *Humanoid* a *New Agent* (žltý rámček). Plochy, na ktorých je *NavMesh* vytvorený, sú v Unity Editore vyznačené modrou farbou.



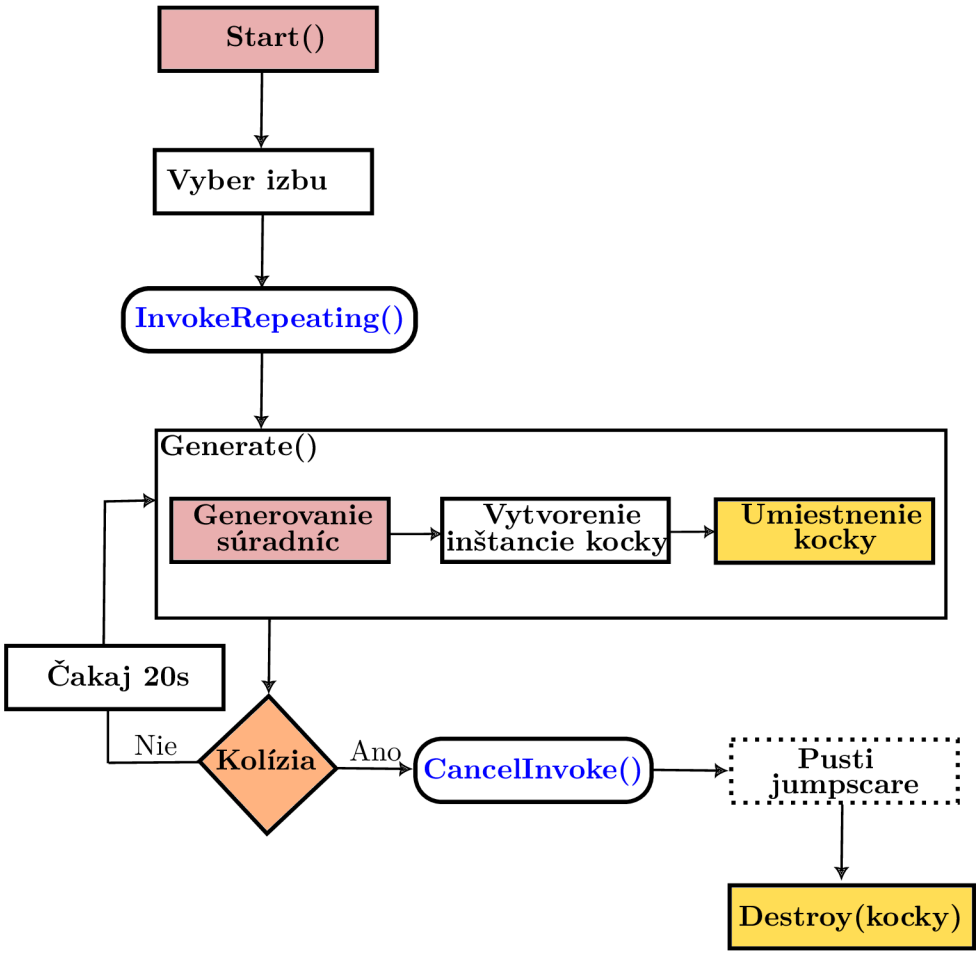
Obrázek 6.24: Algoritmus umiestnenia nepriateľov do labyrintu. Ako prvé algoritmus vytvorí na plochách, kde sa môže nepriateľ pohybovať *NavMesh*. Keď je vytvorený, môže sa vybrať miesto, kde sa nepriateľ umiestni a následne sa môže na to miesto umiestniť. Spolu s nepriateľom sa umiestni aj bod, ktorý bude prenasledovať. Keď je nepriateľ aj s bodom umiestnený, vyberie sa vrámci izby nová kocka, kam sa umiestni plechovka a baterka.

6.3 Faktory strachu

Každá dobrá hororová hra, by mala hráča aspoň trochu vystrašiť. Preto je implementovaných niekoľko mechaník, ktoré sa snažia v hráčovi tento pocit vzbudiť. Hra tieňov, spomenutá v návrhu, nebola kvôli nedostatku času nakoniec implementovaná.

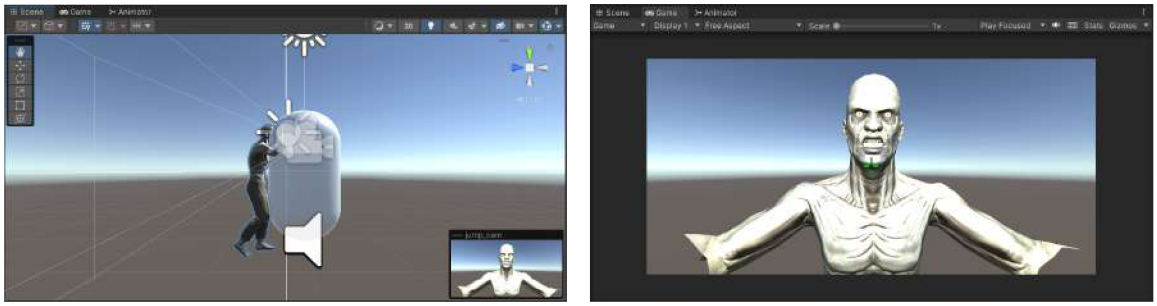
6.3.1 Jumpscare

Jumpscare je založený na generovaní neviditeľných kociek na náhodných pozíciach, do ktorých keď hráč narazí, vyskočí naňho zombie. Základom tejto mechaniky je kocka vytvorená v Unity, ktorá má vypnutý *MeshRenderer*, takže ju v scéne nemožno vidieť. Zároveň obsahuje vlastnosť *BoxCollider*, pomocou ktorej možno rozoznať, či došlo ku kolízií s hráčom alebo nie. Táto kocka je potom umiestňovaná vo vybranej izbe v labyrinte (algoritmus 6.25).

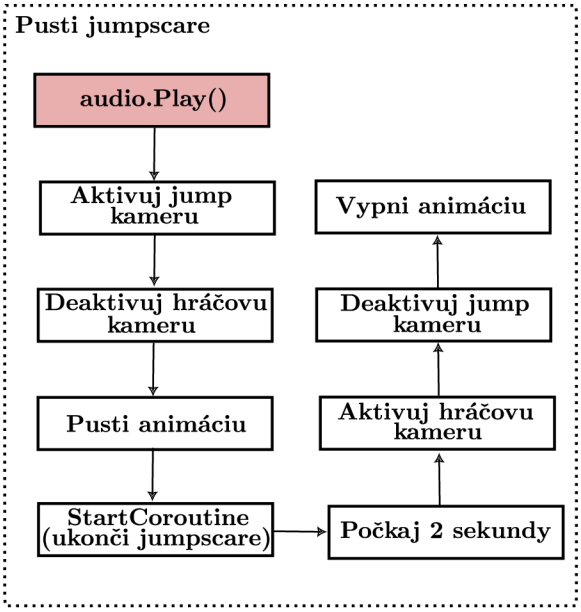


Obrázek 6.25: Obrázok znázorňuje algoritmus generovania náhodných neviditeľných kociek a akým spôsobom kocky reagujú na kolíziu s hráčom. Po spustení sa za pomoci metódy *InvokeRepeating*, ktorá opakovane každých 20 sekúnd umiestni kocku na náhodnom mieste vo vybranej izbe. Z izby sa vyberú náhodné súradnice. Následne na to sa vytvorí klon kocky, ktorá spúšťa jumpscare a tento klon sa umiestni do miestnosti. Počas postupnej generácia dochádza aj ku kontrole, či nejaká kocka neprišla do kolízie s hráčom. Ak áno, zruší sa volanie metódy, ktorá generuje kocky a všetky doposiaľ vytvorené kocky sa zmažú.

Hráč má k sebe pripojenú ďalšiu kameru. V zornom poli tejto kamery je umiestnená zombie (obrázok 6.26). Kamera je celý čas neaktívna až do momentu, kým dôjde ku kolízií s hráčom (obrázok 6.27). Okrem kamery je vrámci *Canvasu* umiestnená animácia. Animácia bola nahraná v Unity. Pozadie *Canvasu* sa zachytilo v rôznych momentoch – keď je úplne čierny, keď je čierna menej intenzívna a keď je jej intenzita na 0. Tento snímok sa postupne prehráva znovu a znovu a dáva dojem, že obrazovka bliká.



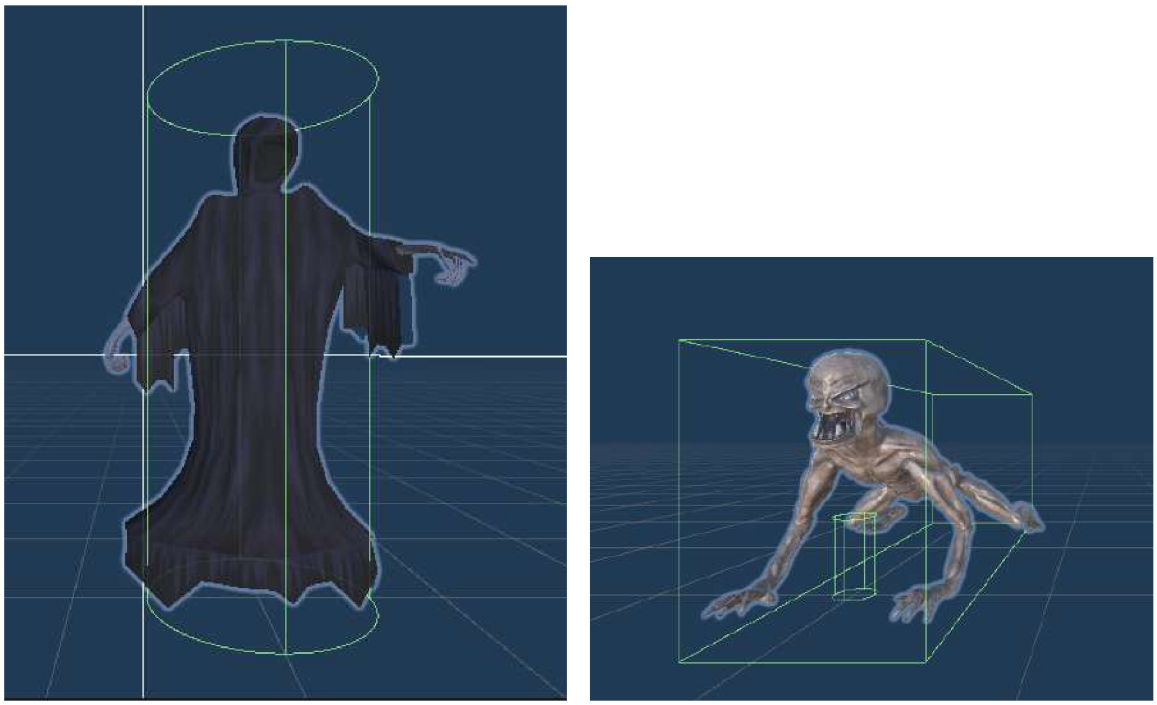
Obrázek 6.26: Hráčova druhá kamera má v zornom poli umiestnenú zombie. Zombie je umiestnená tak blízko ku kamere, že hráč nadobúda pocit, že naňho útočí.



Obrázek 6.27: Algoritmus, ktorý popisuje proces prehrania jumpscaru. V momente, keď dôjde ku kolízií s kockou, pustí sa desivý zvuk, aktivuje sa hráčova druhá kamera a animácia. Po uplynutí dvoch sekúnd sa kamera so zombie vypne a hráč môže pokračovať v hre ďalej.

6.3.2 Nepriatelia

Cieľom nepriateľa je zabiť hráča a neumožniť mu dostať sa z labyrintu von. V hre sa vyskytujú dva typy nepriateľov – duch a crawler (obrázok 6.28). Aby sa mohli pohybovať po vytvorenej navigačnej sieti, majú obaja priradený komponent *NavMeshAgent*. Vrámci tohto komponentu je potom nastavené, po ktorej navigačnej sieti sa má ktorý nepriateľ pohybovať.



Obrázek 6.28: Modely nepriateľov, ktoré boli použité vo výslednej aplikácií. Na ľavej strane je duch a na pravej crawler.

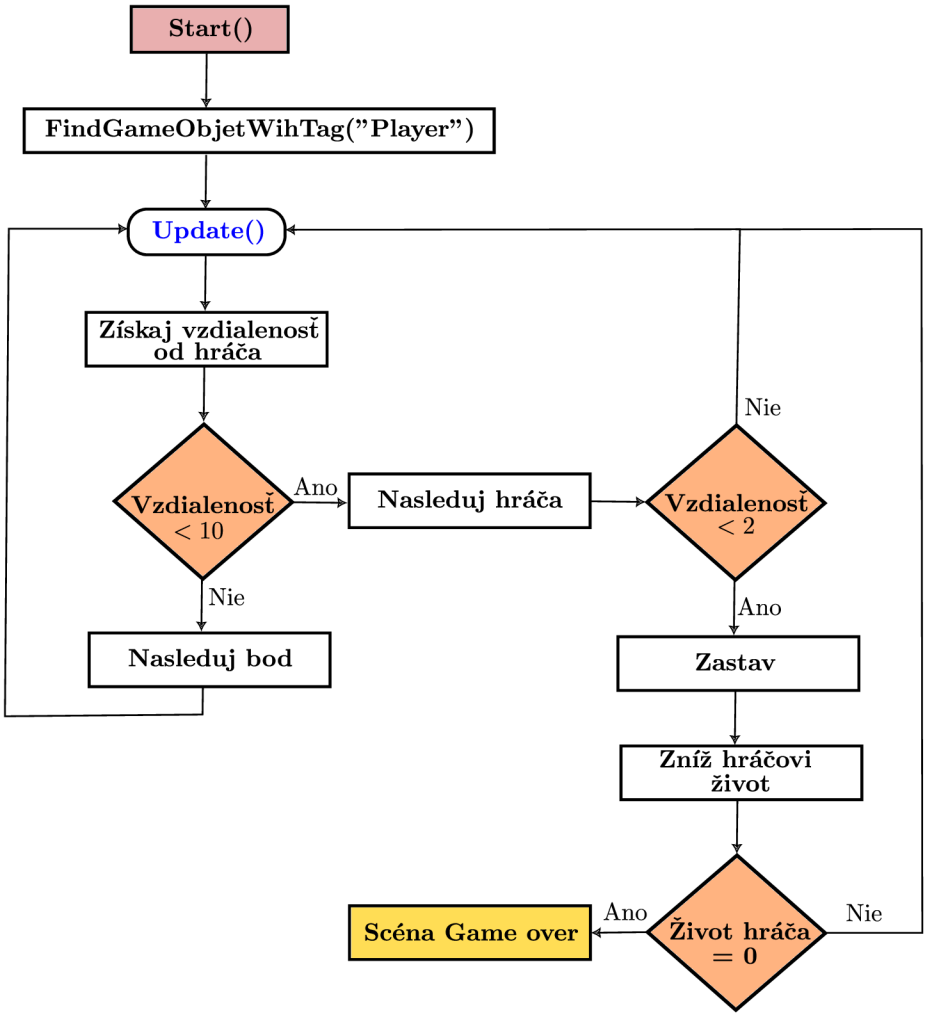
Oba druhy nepriateľov, pokiaľ sú dostatočne blízko, útočia na hráča. Útok nepriateľa sa vykonáva len v prípade, ak ubehne aspoň päť sekúnd od posledného útoku. To možno dosiahnuť výpočtom podľa podmienky 6.1. Zabráni sa tomu, aby nepriateľ neustále zasahoval a hráč mal možnosť utiecť. Po každom útoku sa aktualizuje textové pole s aktuálnym zdravím hráča. Ak zdravie klesne na nulu, načíta sa scéna *Game over* a hra končí.

$$Time.time - last \geq calmDown \tag{6.1}$$

V podmienke 6.1 je $Time.time$ aktuálny čas v sekundách Unity od začiatku hry, $last$ je čas posledného útoku nepriateľa a $calmDown$ je konštanta, ktorá určuje interval, počas ktorého má nepriateľ čakať, kým znovu zaútočí.

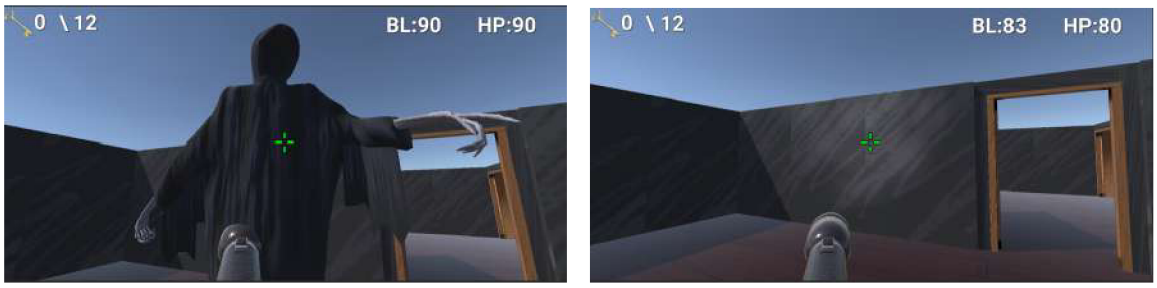
Duch

Duch sa pohybuje voľne po labyrinte a nedostáva sa do kolízie so žiadnymi prekážkami. Hráča môže teda prekvapiť kdekoľvek a môže zaútočiť kedykoľvek. Algoritmus jeho pohybu je znázornený na obrázku 6.29.

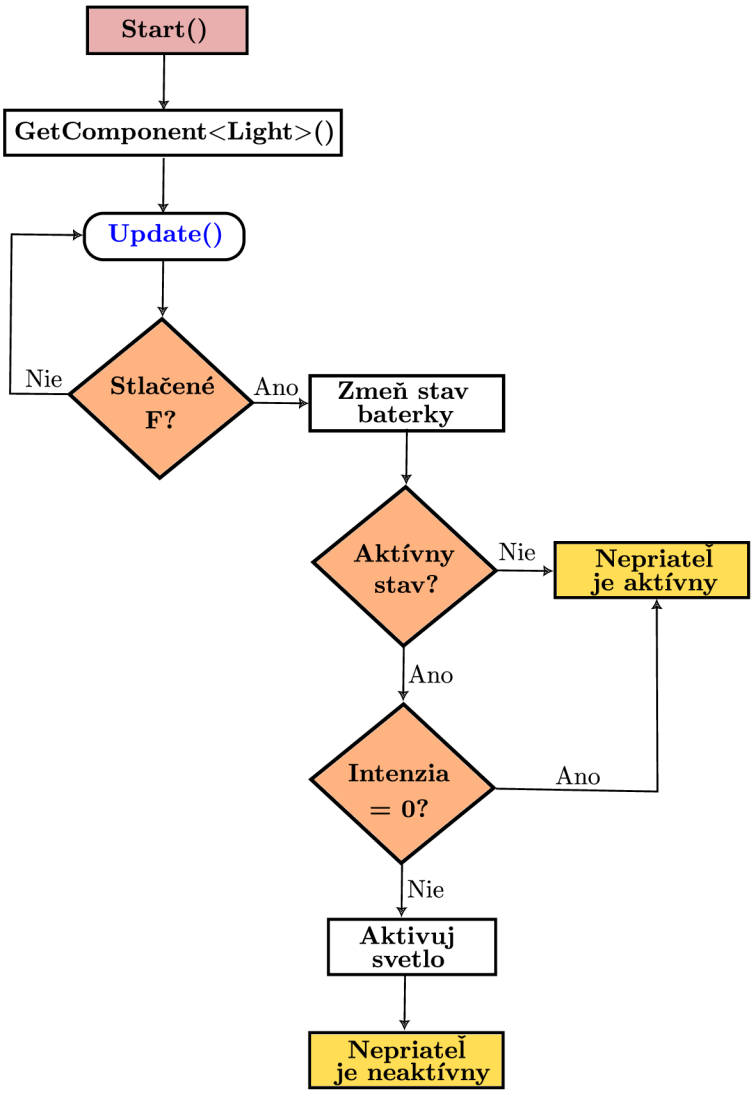


Obrázek 6.29: Algoritmus pohybu ducha po labyrinte. Na začiatku si algoritmus vyhľadá hráča. Následne každý snímok sleduje, aká je vzdialenosť medzi duchom a hráčom. Pokiaľ je vzdialenosť menšia ako 10, duch začne hráča prenasledovať až kým sa hráč znovu nevzdiali. Pokiaľ vzdialenosť medzi ním a hráčom dosiahne hodnoty 2, duch zastaví a začne útočiť na hráča. Hráčovi sa začne znižovať úroveň života. Ak hodnota hráčovho života klesne na 0, načíta sa *GameOver* scéna a hra je ukončená.

Pokiaľ ma hráč v ruke baterku a zasvieti ňou, duch zmizne a hráč môže bez obáv prejsť (obrázok 6.30). Celý tento algoritmus je znázornený na obrázku 6.31.



Obrázek 6.30: Výsledná implementácia svietiacej baterky v Unity. Keď je baterka vypnutá, duch útočí na hráča. V momente, keď hráč zapne baterku duch zmizne.



Obrázek 6.31: Obrázok znázorňuje algoritmus vypnutia/zapnutia baterky. Baterku je možné zapnúť/vypnúť pomocou stlačenia "F". Po stlačení sa zmení stav baterky. Ak je aktuálny stav neaktívny, ducha je možné vidieť. Ak je stav baterky aktívny, ale jej intenzita dosahuje hodnoty 0, nepriateľa je tiež možné vidieť. V prípade že hodnota intenzity je vyššia ako 0, aktivuje sa svetlo a nepriateľ zmizne z hernej scény.

Baterka má ale svoju kapacitu. Táto kapacita sa znižuje s časom, ktorý svieti (obrázok 6.32). Keď dosiahne hodnoty 0, baterka je vybitá a nesvieti. Baterka má ako potomka svetlo, ktoré obsahuje komponentu *Light*. Jednou z vlastností tohto herného objektu je intenzita. Tú je možné meniť v čase a dosiahnuť tak, že svetlo bude postupne zhasínať až bude baterka úplne vybitá. Intenzita svetla sa znižuje lineárne v závislosti od času a od pôvodnej intenzity svetla. Nová intenzita sa počíta v každom novom snímku podľa vzorca 6.2:

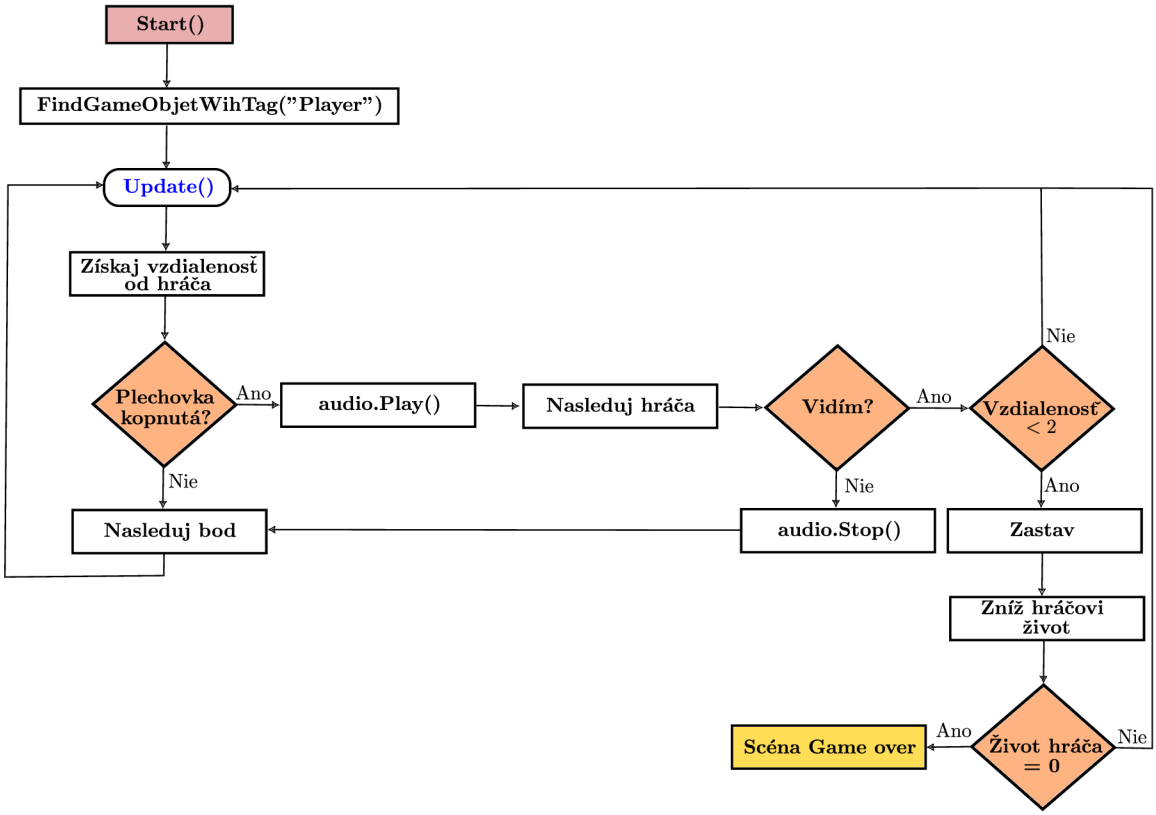
$$I_{k+1} = I_k - \frac{I_k}{V} \cdot \delta t; \tag{6.2}$$

V rovnici 6.2 znamená symbol V celkový čas, ktorý baterka vydrží svietiť, I_k je aktuálna intenzita svetla, I_{k+1} je nová intenzita svetla a δt udáva čas, ktorý ubehol od posledného snímku v sekundách.

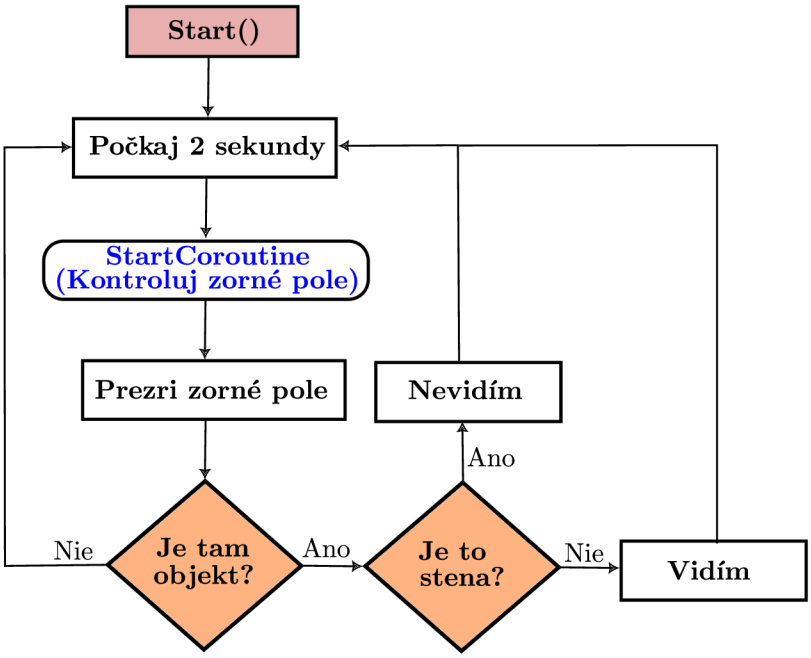
Obrázek 6.32: Na začiatku, keď je baterka vypnutá je jej životnosť 100. V momente, keď sa zapne, začne jej výdrž postupne klesať.

6.3.3 Crawler

Táto príšera nasleduje náhodný bod až do chvíle, keď dôjde ku kolízii hráča s plechovkou (obrázok 6.33). Dôležitou súčasťou algoritmu je *FieldOfView* (algoritmus 6.34). Crawler má svoje zorné pole a má určený maximálny dosah dohľadu, kam dovidí (v akej vzdialenosti je schopný detekovať objekty).



Obrázek 6.33: Algoritmus popisuje spôsob, akým sa crawler pohybuje po labyrinte a ako reaguje na hráča. Algoritmus si ako prvý nájde v scéne hráča. Následne každý snímok kontroluje, či nedošlo ku kolízii hráča s plechovkou. Pokiaľ áno, plechovka bola kopnutá. Po kopnutí sa spustí desivé audio, ktoré symbolizuje, že hráča začal prenasledovať nepriateľ. Pokiaľ crawlerovi nič nestojí v jeho zornom poli a je dostatočnej vzdialenosti od hráča, začne útočiť. Pokiaľ narazí na prekážku, prenasledovanie sa ukončí, hudba sa vypne a crawler si ďalej nasleduje svoj bod.



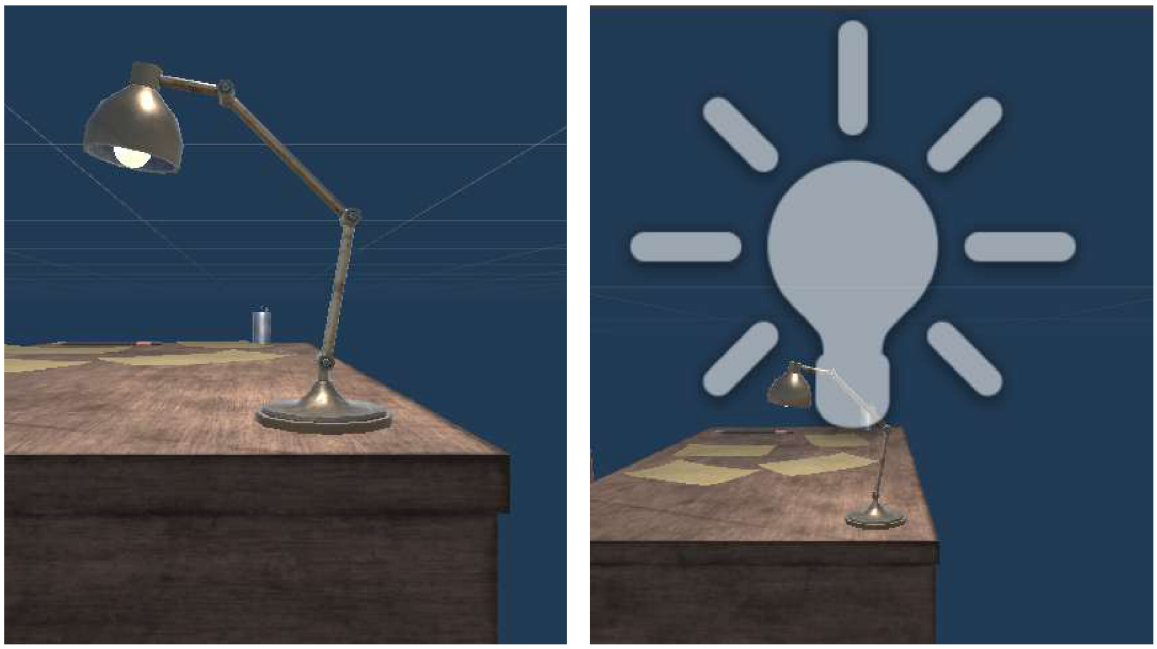
Obrázek 6.34: Obrázok algoritmu popisuje, akým spôsobom crawler kontroluje, či je pred hráčom prekážka alebo nie. Každé dve sekundy sa kontinuálne spúšťa korutina, ktorá vyvoláva detekciu prekážok v zornom poli nepriateľa. Algoritmus najskôr získa všetky objekty, ktoré sa nachádzajú v okruhu nepriateľa. Ak nie sú žiadne objekty v okruhu, nepriateľ hráča vidí. Ak sa však v okruhu nachádza nejaký objekt (hráč alebo iný), skontroluje sa, či to nie je stena. Pokiaľ áno, hráča nie je možné vidieť. Na detekciu prekážok medzi hráčom a nepriateľom sa používa funkcia *Raycast*. Táto funkcia vytvára lúč (ray) z aktuálnej pozície nepriateľa v smere hráča a testuje, či narazí na nejakú prekážku, ktorá by bránila vidieť hráča.

6.3.4 Prostredie

Aby atmosféra hry bola ešte strašidelnejšia, boli implementované dve jednoduché vizuálne mechaniky – blikajúca lampa a zvuky, rozmiestnené rôzne po labyrinte.

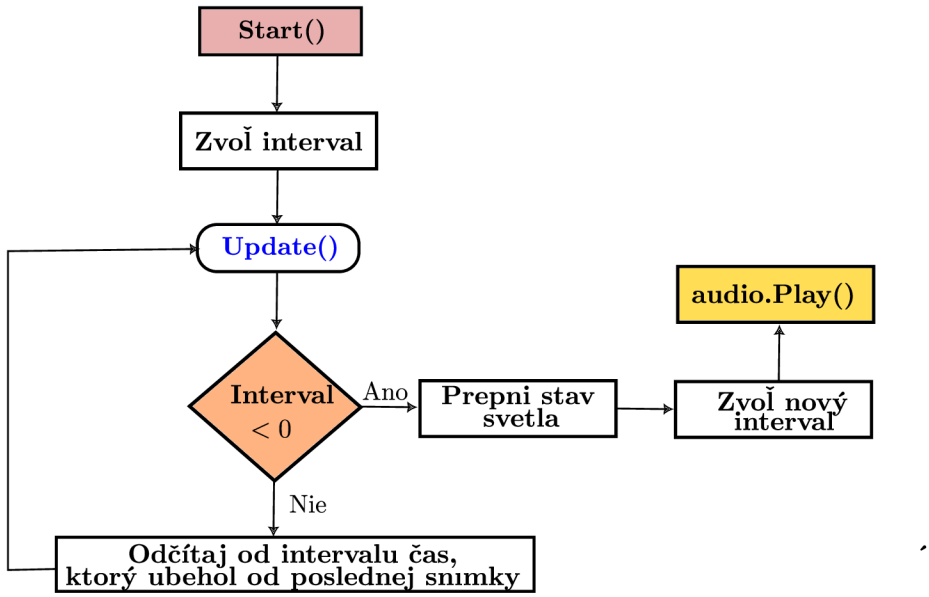
Blikajúce svetlo

Výsledná hra má temné prostredie. Z tohto dôvodu, je dobré dotvoriť atmosféru svetlom. Na niektorých stoloch v miestnostiach sa nachádza lampa, ktorá náhodne preblikáva (obrázok 6.35). Algoritmus je znázornený na obrázku 6.36.



(a) (b)

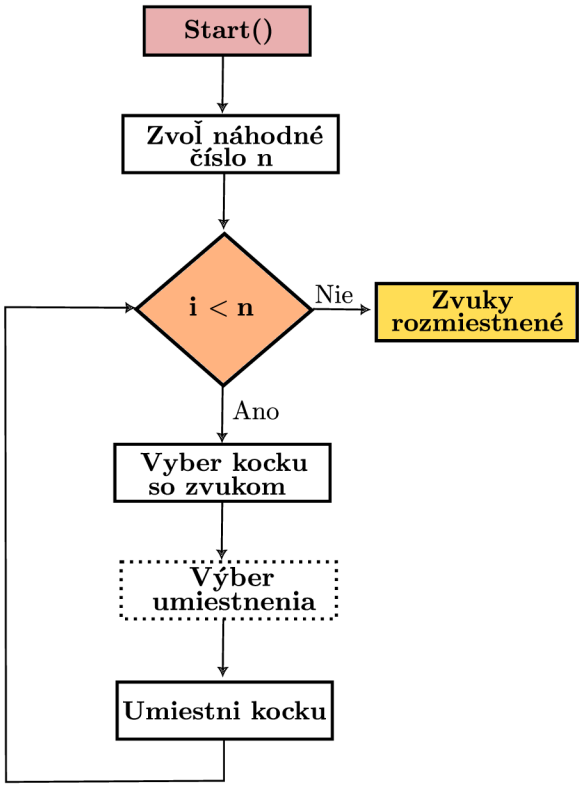
Obrázek 6.35: Lampa na stole (obrázok a) má umiestnené nad sebou svetlo (obrázok b), ktoré sa v priebehu hry aktivuje, deaktivuje. Tým, že svetlo stále mení svoj stav a raz sa v scéne nachádza a raz nie, vytvára dojem, ako keby blikalo.



Obrázek 6.36: Algoritmus blikajúceho svetla. Najskôr sa vygeneruje náhodný čas (interval), v ktorom bude na začiatku lampa preblikávať. Následne, sa od tohto intervalu bude postupne odčítavať čas, ktorý ubehol od poslednej snímky. Keď je čas menší ako 0, dôjde k zmene stavu svetla (aktívne, neaktívne) a zvolí sa nový interval, v ktorom sa bude čakať, kým dôjde k zmene stavu svetla. S prepnutím stavu sa spustí zvuk poškodenej lampy.

Zvuky

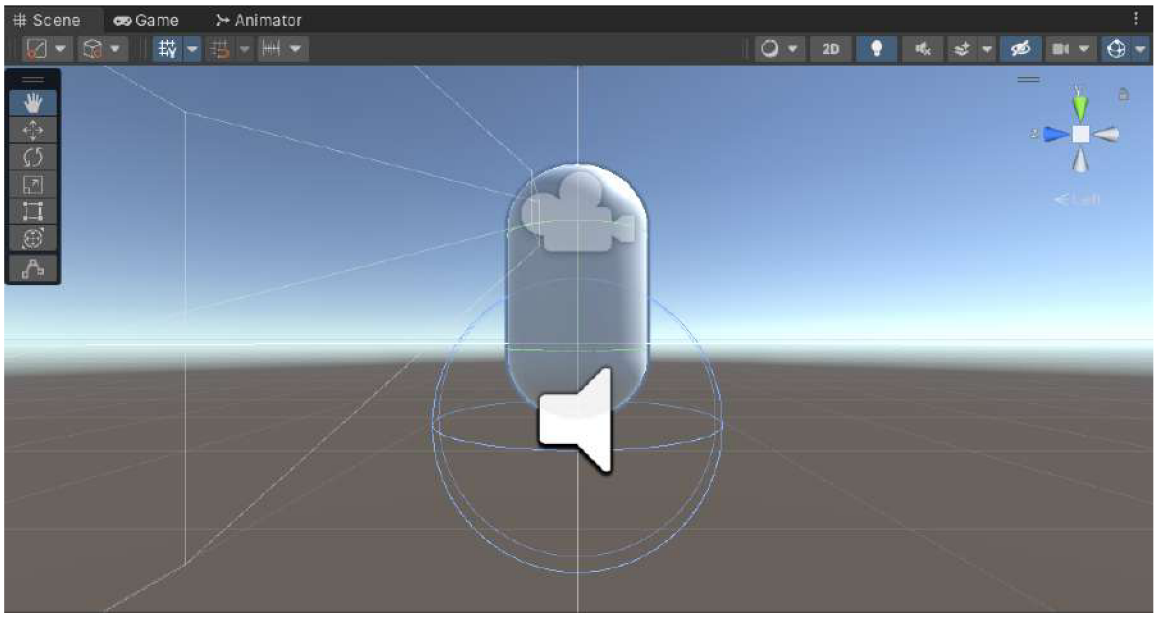
Okrem svetla sú v hre rozmiestnené rôzne hrôzostrašné zvuky. Zvukov je niekoľko druhov a vždy sa v labyrinte rozmiestnia len niektoré (obrázok 6.37). Zvuky sú umiestnené na prázdnych herných objektoch, takže ich nie je možné v hre vidieť a ani sa dostať s nimi do kolízie. Všetky zvuky sú nastavené ako 3D, čo znamená, že ich je možné vnímať z rôznych smerov na základe polohy poslucháča. Zároveň všetky majú nastavenú maximálnu a minimálnu vzdialenosť. Keď sa hráč dostane do maximálnej vzdialenosti od objektu so zvukom, zvuk počuť veľmi potichu. Čím bude bližšie, tým bude zvuk hlasnejší. Keď sa dostane mimo dosahu zvuku, nepočuť ho vôbec.



Obrázek 6.37: Algoritmus rozmiestnenia náhodných zvukov po labyrinte. Na začiatku sa zvolí počet, koľko zvukov sa má do hry umiestniť. Algoritmus potom cyklicky prechádza cez toto číslo. Vždy náhodne zvolí jeden zvuk a umiestni ho na miesto vybrané podľa algoritmu 6.18.

6.4 Hráč

Hráč (First Person Controller) je prevzatý z Unity asset storu, kde boli k nemu už priložené scripty pre chôdzu, beh, zakrádanie a skákanie (obrázok 6.38). Boli k nemu ale priradené ďalšie skripty ako napríklad scripty, vďaka ktorým môže hráč pracovať s inventárom. Tiež je naňho naviazaný jednoduchý script, ktorý pri zapnutí hry skryje a uzamkne kurzor myši (obrázok 6.39). Hráč je potom schopný ovládať hru len pomocou kláves a kameru otáčať pomocou myši.

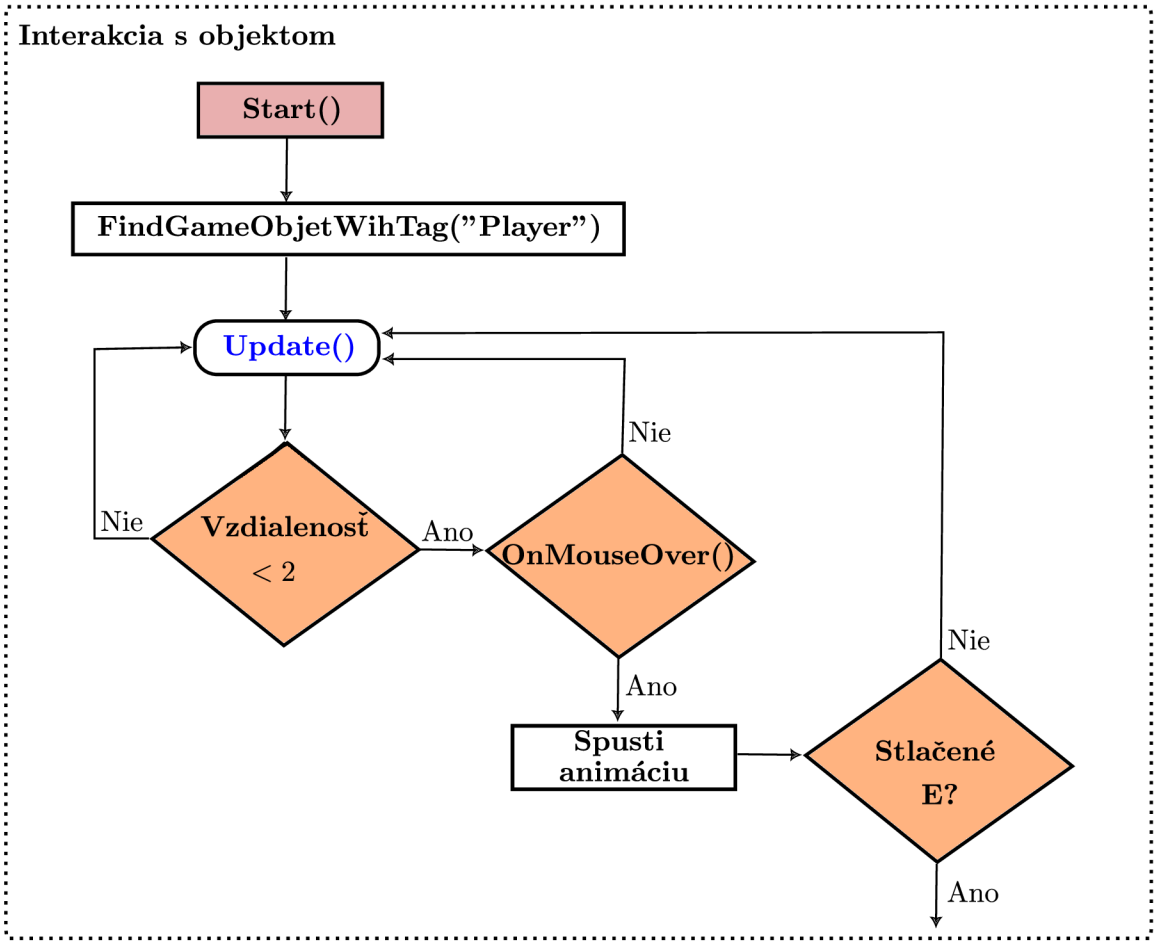


Obrázek 6.38: Model hráča(First Person Controller), ktorý sa používa na pohyb po labyrinte.

Obrázek 6.39: Jednoduchý algoritmus pre kurzor. Po zapnutí hry dôjde k jeho zneviditeľneniu a deaktivovaniu.

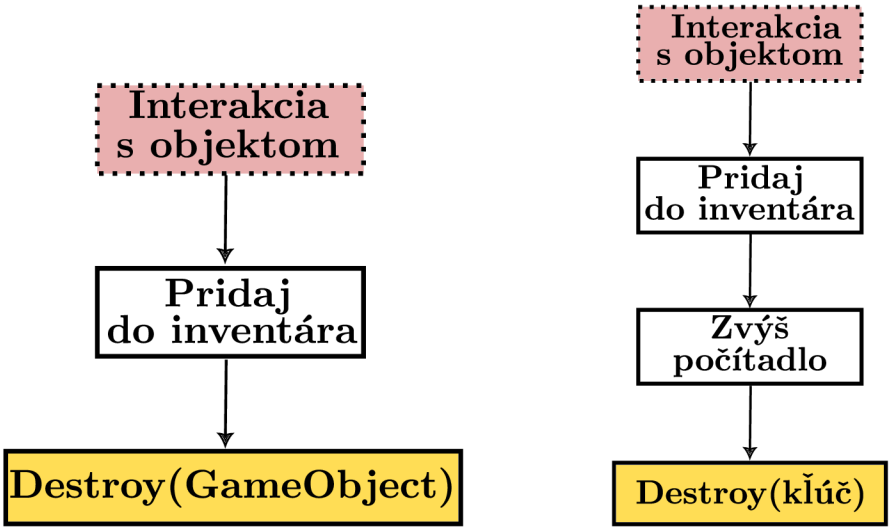
6.4.1 Zbieranie predmetov

Či už hráč zbiera kľúče, baterky alebo lekárničky, základný algoritmus je vždy rovnaký. Každý objekt má priradený skript, ktorý v každom novom snímku kontroluje, v akej vzdialenosti je hráč, či je zameraný na objekt myšou alebo či stlačil "E"(algoritmus 6.40).



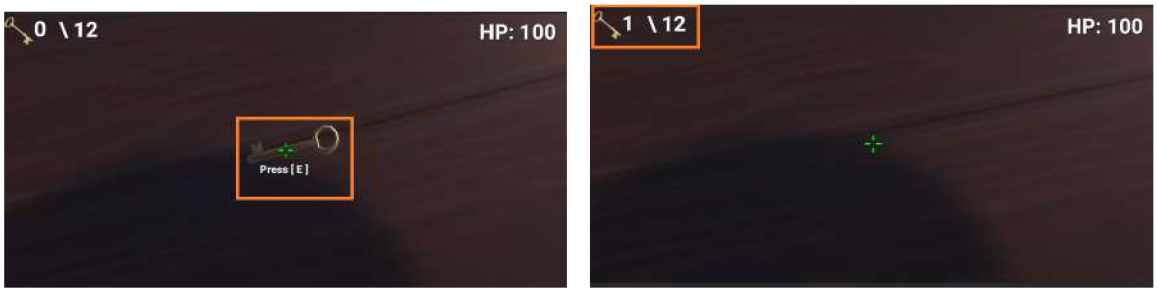
Obrázek 6.40: Základný algoritmus, ktorý je spoločný pre zbieranie všetkých predmetov a otváranie dverí. Na začiatku si algoritmus nájde hráča a počas celého chodu hry sleduje jeho pohyb. Ak sa dostane do určite vzdialenosti, je možné s objektom pracovať. Pokiaľ je dostatočne blízko a zameria kurzor na objekt, spustí sa animácia, ktorá simuluje pulzujúci kurzor. Pokiaľ hráč stlačí "E", vykoná sa príslušná akcia.

Či už hráč zbiera lekárničky, tužkové batérie alebo kľúče, algoritmus je až na pár drobností rovnaký (algoritmus 6.41). Počet kľúčov sa hráčovi ukazuje ešte aj na obrazovke, vďaka čomu má dobrý prehľad o ich počte (obrázok 6.42).



(a) (b)

Obrázek 6.41: Obrázky znázorňujú algoritmus zdvihnutia predmetov zo zeme. V prípade zbierania predmetov sa hráčovi po stlačení "E"pridá objekt do inventára a fyzicky je odstránený z hernej scény. V prípade kľúčov, dôjde aj k zvýšeniu počítadla, čím sa aj hráčovi zmení informácia o počte kľúčov na obrazovke (obrázok b).



(a) (b)

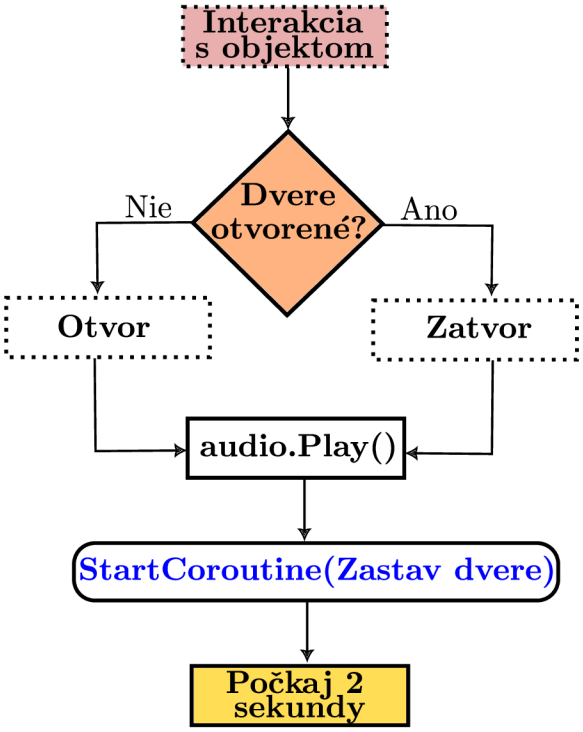
Obrázek 6.42: Pri umiestnení kurzora na kľúč sa hráčovi zobrazí výzva na zdvihnutie kľúča (obrázok a). Po jeho zdvihnutí zmizne objekt zo scény a zvýši sa tak číslo na počítadle kľúčov na obrazovke (obrázok b)

6.4.2 Otváranie dverí

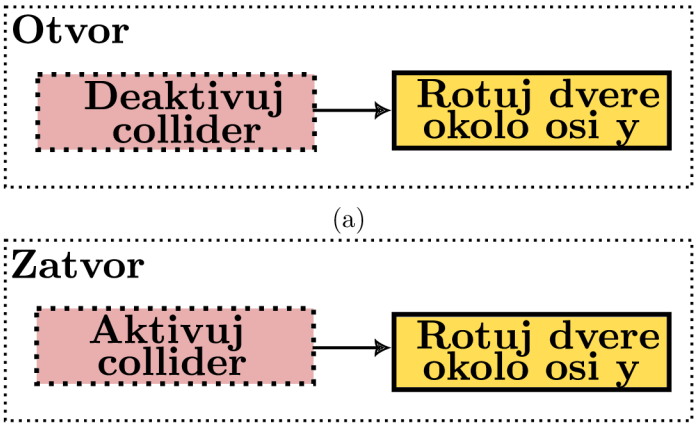
Neide o žiadnu animáciu. Celý posun je zabezpečený pomocou algoritmu 6.43. Podobne ako pri zbieraní objektov, aj tu každý herný objekt dverí sleduje a kontroluje vzdialenosť hráča (algoritmus 6.40). Základný princíp pre otváranie dverí bol inšpirovaný tutoriálom dostupným na YouTube [2].

Každé dvere majú priradený komponent *RigidBody*. Vďaka tomuto komponentu, je dvere možné otočiť okolo hociktorej osi a docieliť tak dojem, že sa dvere otvárajú (obrázok 6.44) . Keďže tento objekt neumožňuje mať na dverách aktívny *Collider*, bola vytvorená a umiestnená pred každé dvere kocka, ktorá zabraňuje hráčovi prejsť cez dvere, keď sú zatvorené. Hráč tak dvere musí otvoriť, aby sa dostal ďalej (algoritmus 6.45).

[2]https://www.youtube.com/watch?v=8d1OhNQNSuY

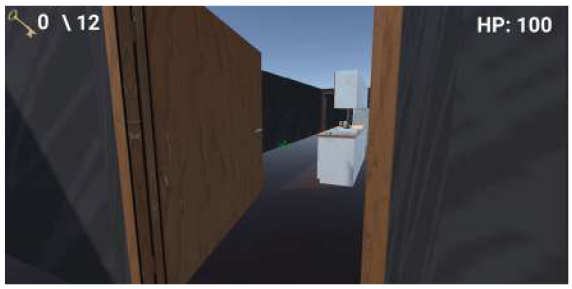


Obrázek 6.43: Alogritmus si uchováva stav dverí. Podľa toho, aký je ich aktuálny stav, dvere buď otvorí alebo zatvorí. Spolu so zatváraním dverí sa prehrá aj audio, ktoré simuluje zvuk zatvárania starých dverí. Následne je privolaná korutina, ktorá zabezpečí, aby sa dvere neotáčali okolo osi donekonečna, ale aby sa po posune zastavili.



Obrázek 6.44: Dvere rotujú okolo osi y. V prípade otvárania dverí sa ich pozícia posunie v smere osi y o -2, v prípade zatvorenia o 2. Predtým, než sa dvere otvoria (obrázok a) sa deaktivuje kocka, ktorá slúži ako zábrana, aby mohol hráč prejsť cez dvere. V prípade zatvorenia sa táto kocka znovu aktivuje

(a)



(b)

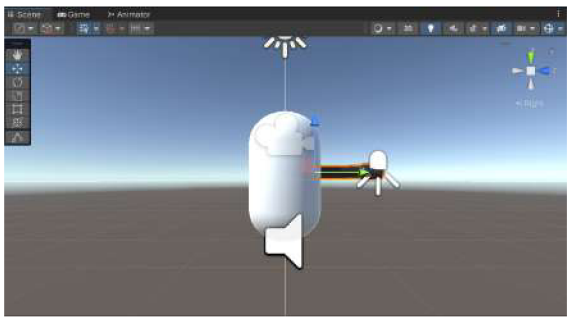
Obrázek 6.45: Pri zameraní kurzora na dvere sa hráčovi objaví na obrazovke výzva k otvoreniu dverí (obrázok a). Po stlačení "E"sa dvere otvoria a hráč tak môže prejsť do ďalšej miestnosti (obrázok b)

Koncové dvere

Špeciálnym typom dverí sú dvere, ktoré slúžia ako únik z labyrintu. Tie možno otvoriť, až keď je splnená podmienka a počítadlo nazbieraných kľúčov dosiahne hodnoty 12. Tieto dvere sa však neotvoria, ale načíta sa výherná scéna *YouWin*.

6.4.3 Baterka

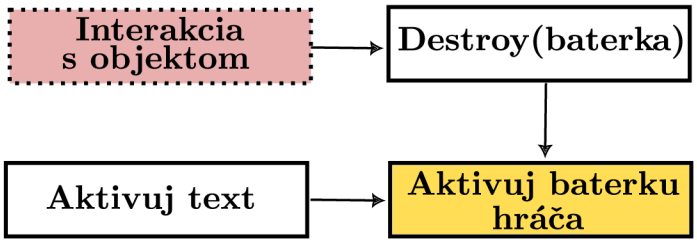
K hráčovi je priradený herný objekt baterky. Poloha baterky je nastavená tak, aby ju hráč videl v zornom poli kamery (obrázok 6.46). Baterka potom pôsobí, ako keby ju hráč držal v ruke. Na začiatku je baterka neaktívna a nemožno ju v hre vidieť. Po tom, ako hráč zdvihne zo zeme ležiacu baterku, tak dôjde k jej aktivácií (obrázok 6.47).



(a)

(b)

Obrázek 6.46: Baterka je pri hráčovi umiestnená tak, aby bola pri spustení hry v zábere kamery (obrázok b).



Obrázek 6.47: Obrázok, ktorý popisuje algoritmus zdvihnutia baterky zo scény. Začiatok algoritmu je rovnaký ako pri zbieraní predmetov či otváraní dverí (obrázok 6.40). V tomto prípade sa po stlačení "E"herný objekt baterky vymaže zo scény a dôjde k aktivácií baterky, ktorú má hráč pripojenú k sebe. Zároveň sa na obrazovke aktivuje text, ktorý informuje hráča o tom, koľko má baterka ešte kapacity.

6.5 Inventár

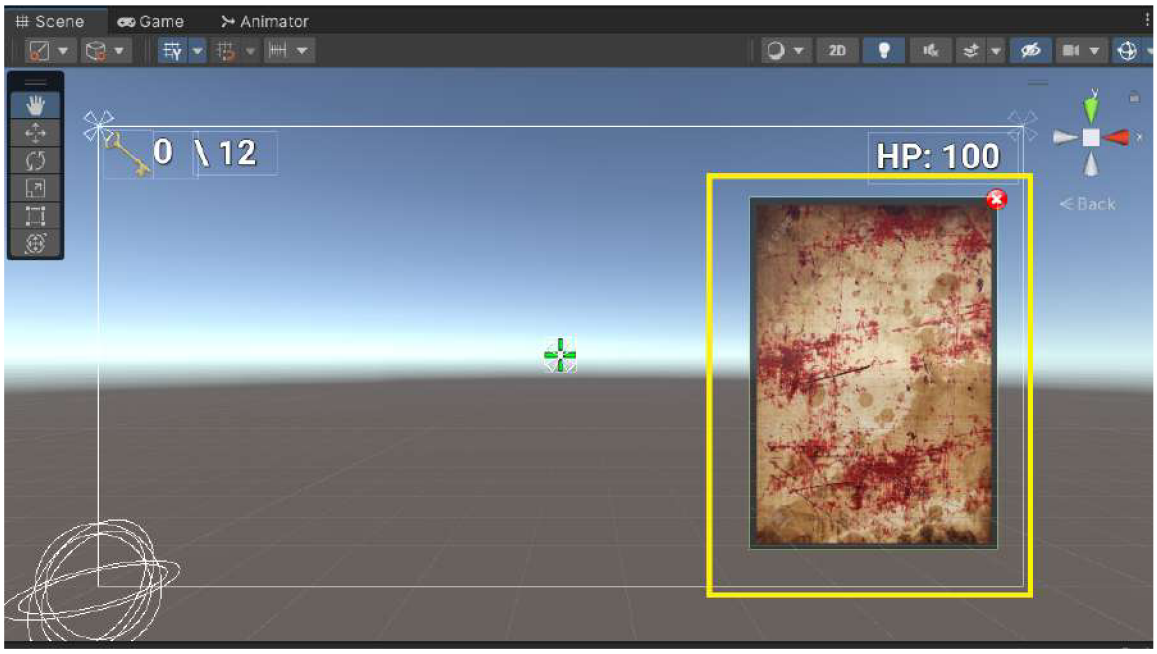
Hráč má v hre možnosť zbierať predmety do inventára (obrázok 6.48). Pre inšpiráciu pri tvorbe inventára bol použitý tutoriál dostupný na YouTube [3].

(a) (b)

Obrázek 6.48: Keď hráč nájde v hre objekt a zameria naň kurzor, zobrazí sa informačná hláška, ktorá ho vyzýva k zdvihnutiu predmetu (obrázok a). Po tom čo hráč objekt zo zeme zdvihne, zmaže sa z hernej scény a zobrazí sa v inventári (obrázok b)

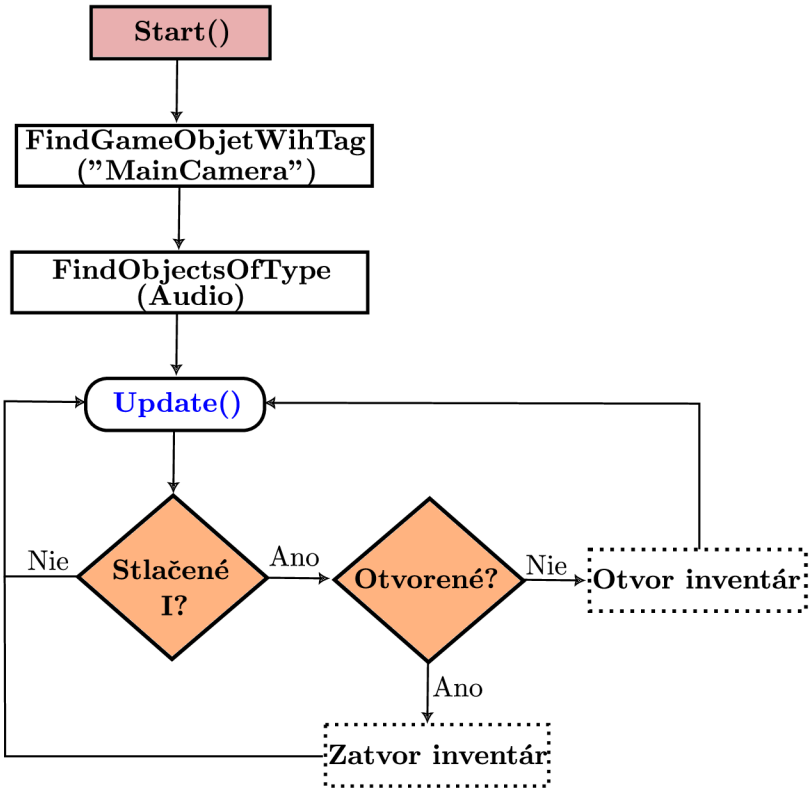
Inventár je reprezentovaný ako 2D objekt *ScrolledView*, ktorý je umiestnený vrámci *Canvasu* (obrázok 6.49).

[3] https://www.youtube.com/watch?v=AoD_F1fSFFg



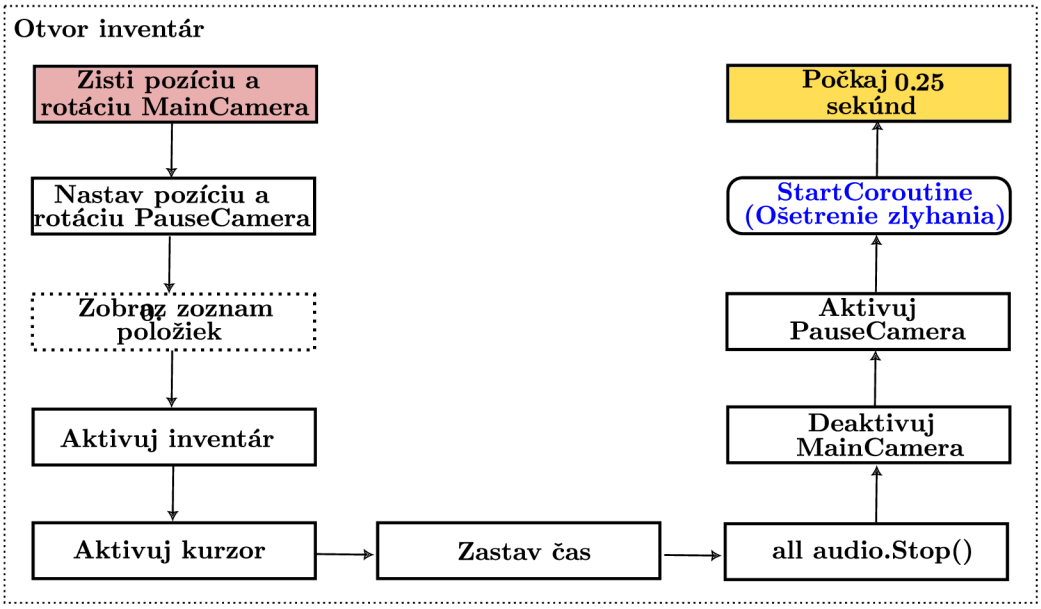
Obrázek 6.49: Vzhľad inventára. Inventár je možné zobraziť tlačidlom I, inak je neaktívny. Zatvoriť ho je možné buď opätovným stlačením "I"alebo tlačidlom krížik v pravom rohu inventára. Toto tlačidlo má nastavený *OnClick event*, kedy sa po jeho stlačení inventár deaktivuje.

Inventár je aktívny len v prípade, ak si ho hráč sám zobrazí (algoritmus 6.50).

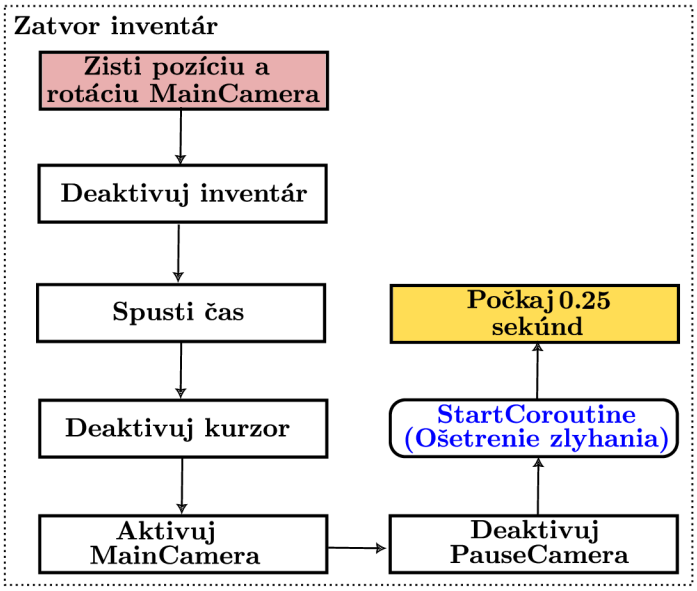


Obrázek 6.50: Algoritmus otvárania inventára. Pokiaľ hráč stlačí "I", zmení sa stav inventára a podľa toho sa hráčovi zobrazí alebo skryje.

Keď je inventár otvorený, chod celej hry sa zastaví. Hráč tak môže pracovať len s inventárom a nehrozí, že naňho niekto zaútočí (algoritmus 6.51). Hráčovi sa zároveň na obrazovke zobrazí kurzor, aby mohol s objektami v inventári pracovať. Po zatvorení sa chod hry vráti do pôvodného stavu (algoritmus 6.52).

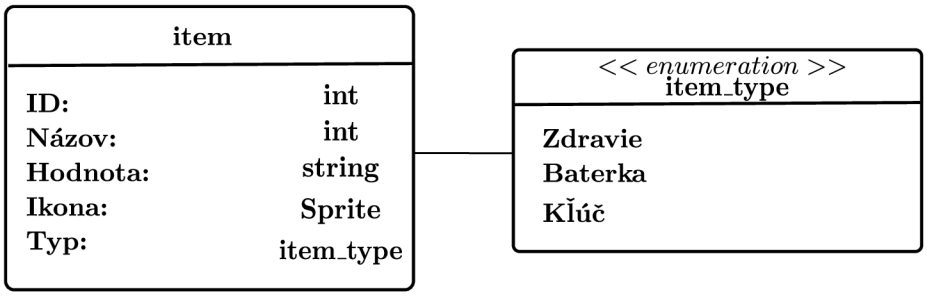


Obrázek 6.51: Algoritmus znázorňuje, k akým všetkým akciám dochádza pri otváraní inventára. Hráč má špeciálnu kameru, ktorá je celý čas, až kým nedôjde k otvoreniu inventára neaktívna. Je to z dôvodu, že aj napriek tomu, že sa zastavil v hre čas, je možné s kamerou hýbať. Aby sa pohľad hráča zastavil na mieste, nastaví sa preto pri otváraní poloha tejto kamery na aktuálnu polohu hráčovej kamery. Následne sa do inventára zobrazí zoznam aktuálnych položiek, ktoré sa v ňom nachádzajú a inventár sa ukáže hráčovi na obrazovke. Čas v hre sa zastaví, rovnako ako aj všetky zvukové efekty. Dôjde k prepnutiu kamier. Následne sa vyvolá korutina, ktorá nasledujúcich 0.25 sekúnd zabráni, aby bolo možné inventár znovu otvoriť/zatvoriť. Táto korutina je implementovaná kvôli lepšiemu chodu hry, lebo môže nastať, že inventár sa zasekne a ostane stále otvorený.

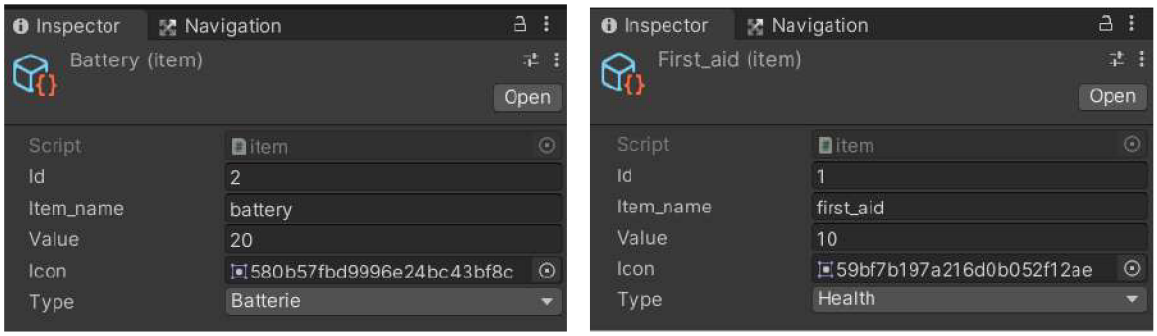


Obrázek 6.52: Algoritmus znázorňuje, aké všetky kroky je potrebné vykonať pri zatváraní inventára. Pri zatváraní inventáru sa deaktivuje vizualizácia inventára, spustí sa znovu čas a deaktivuje sa kurzor. Dôjde k výmene kamier a následne sa spustí korutina, ktorá zabráni opätovnému otvoreniu inventára na ďalších 0.25 sekúnd.

Na správu jednotlivých objektov, ktoré sa v inventári nachádzajú, je vytvorený *Scriptable-Object*, ktorý definuje triedu *item* (obrázok 6.53). Pre všetky zbierané predmety je vytvorený vlastný objekt typu *item*, kde sú definované vlastnosti týchto objektov (obrázok 6.54).



Obrázek 6.53: Trieda, ktorá definuje jednotlivé objekty v inventári. Obsahuje vlastnosti ako je ID, meno, ikona, ktorá sa zobrazí, hodnota objektu a typ. Hodnota objektu predstavuje číslo, ktoré bude pripočítané k životu hráča/baterky.

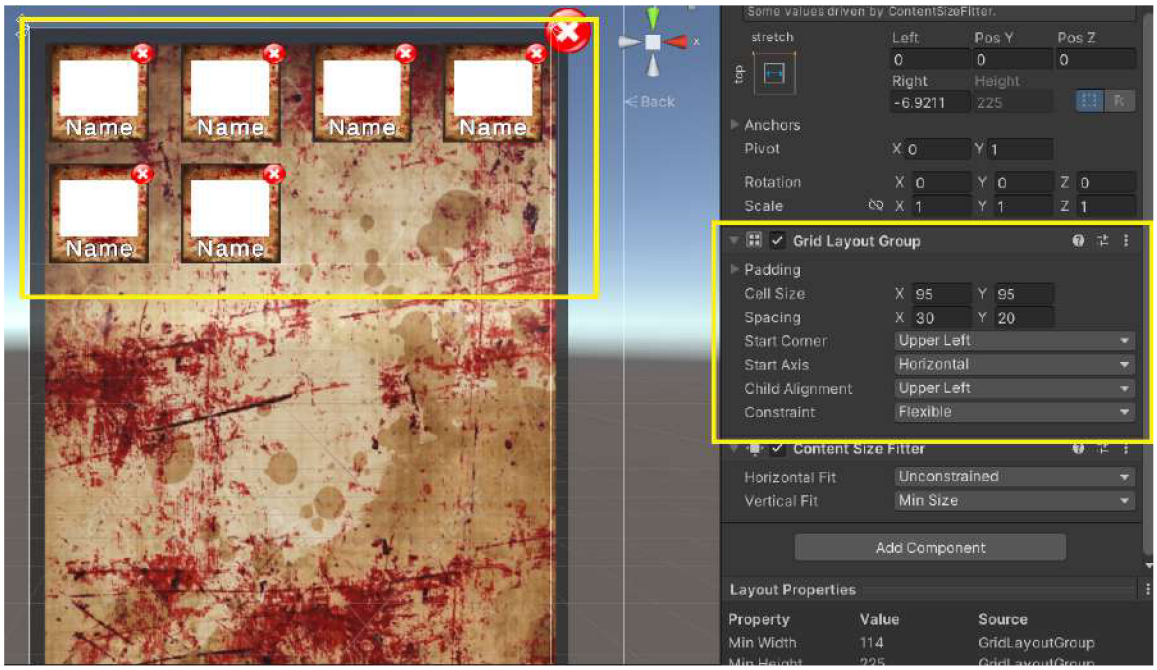


(a) (b)

Obrázek 6.54: Pre každý objekt je vytvorený vlastný item. Podľa druhu objektu sú nastavené jeho vlastnosti (ID, meno, hodnota).

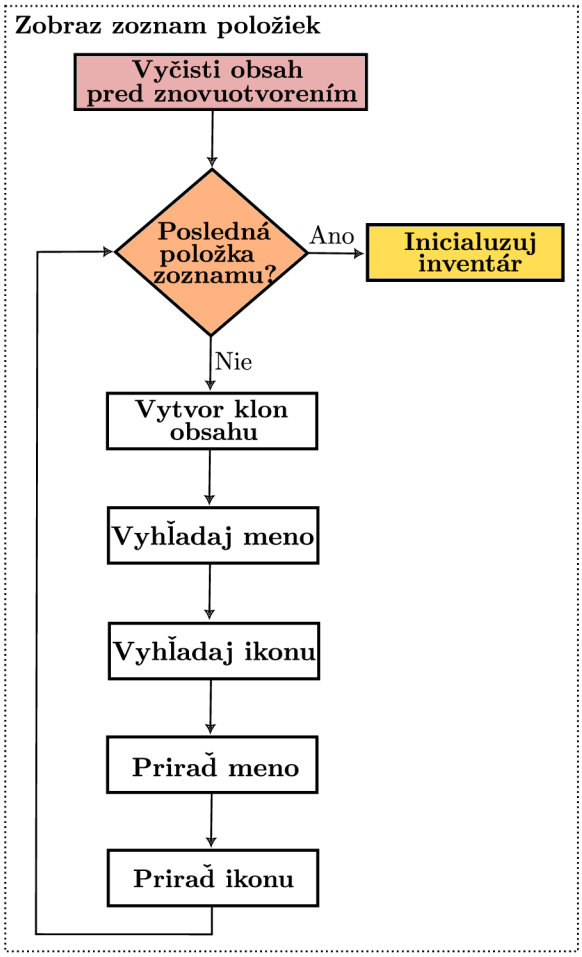
Jednotlivé objekty sa vo vizualizácií inventára zobrazujú ako tlačidlá (buttons). Tlačidlo má priradený *Image*, kde sa zobrazí obrázok daného objektu a *Text* kde sa zobrazí jeho názov (obrázok 6.55) . Aby sa jednotlivé tlačidlá dobre v inventári zobrazovali, je pomocou komponentu *GridLayout* nastavená mriežka, do ktorej sa budú tieto tlačidlá ukladať (obrázok 6.56).

Obrázek 6.55: Tlačidlo, ktoré vizualizuje objekty v inventári. Podľa typu objektu je biele miesto nahradené obrázkom, reprezentujúcim daný objekt a name nahradený názvom daného objektu.



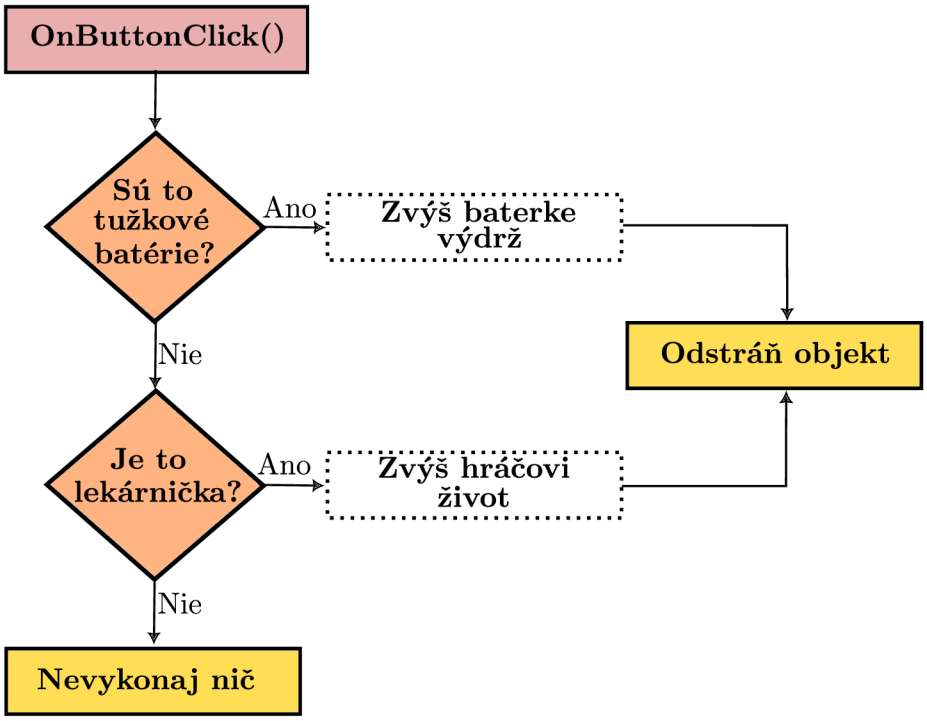
Obrázek 6.56: Jednotlivé tlačidlá sa postupne vedľa seba umiestňujú do mriežky.

Po tom ako sa objekt zdvihne a odstráni zo scény, je pridaný do zoznamu zozbieraných objektov (algoritmus 6.41). Tento zoznam sa použije, pre zobrazenie objektov pri otvorení inventára (algoritmus 6.57).

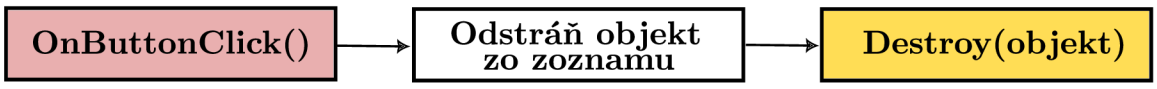


Obrázek 6.57: Algoritmus zobrazovania položiek z inventára vo vizualizácií v užívateľskom rozhraní. Predtým, než sa inventár otvorí, vyčistí sa jeho zobrazený obsah. Keby sa nevyčistil, objekty by sa zobrazovali duplicitne. Po vyčistení sa prechádza zoznamom zozbieraných objektov zo scény. Vytvorí sa klon (inštancia) tlačidla (6.55). Algoritmus si následne nájde referencie na komponent *Image* a *Text*, ktorý toto tlačidlo obsahuje. Na základe triedy, ktorú má objekt priradenú (6.53), si algoritmus vie nájsť obrázok a názov objektu, ktorý sa priradí do komponentov na tlačidle. Následne sa jednotlivé položky inicializujú a sú im priradené dáta, ktoré ich reprezentujú.

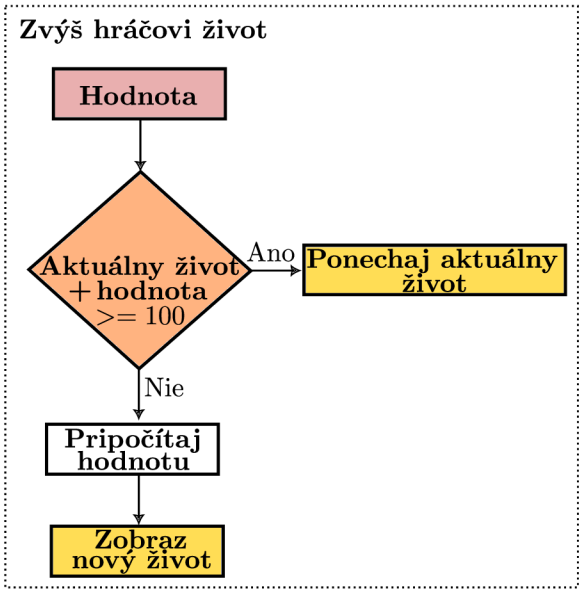
Objekty je v inventári možné buď použiť (algoritmus 6.58) alebo ich z inventára zmazať (algoritmus 6.59). Hráčov život, ako aj životnosť baterky, má maximálnu hodnotu 100, preto pred použitím prebehne kontrola, ktorá zabezpečí, že sa táto hranica po použití objektu neprekročí (algoritmus 6.60). Hráč môže potom v priebehu hry objekty používať podľa potreby (obrázok 6.61).



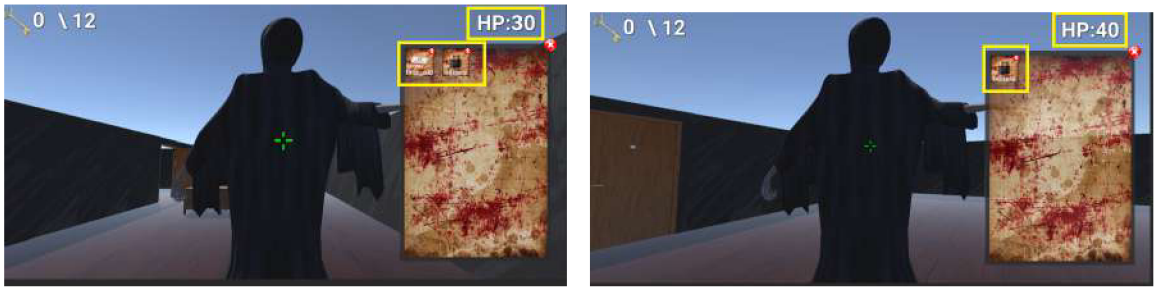
Obrázek 6.58: Obrázok popisuje algoritmus, ktorý sa používa pri použití položky z inventára. Položky v inventári sú tlačidlá, na ktoré je možno kliknúť. Po stlačení tlačidla sa algoritmus pozrie, o aký typ predmetu ide a podľa toho buď zvýši život hráčovi alebo výdrž baterke. Po použití sa objekt zmaže z inventára aj zo zoznamu zozbieraných objektov.



Obrázek 6.59: Obrázok znázorňuje jednoduchý algoritmus, ktorým sa položky mažú z inventára. Tlačidlá reprezentujúce jednotlivé objekty majú na sebe tlačidlo krížik. Keď sa toto tlačidlo stlačí, objekt je odstránený zo zoznamu objektov a aj z inventára.



Obrázek 6.60: Obrázok znázorňuje algoritmus, ktorý kontroluje, či je ešte možné zvýšiť hráčov život alebo nie, Predtým, než sa hodnota k životu hráča alebo výdrži baterky priráta, prebehne kontrola, či nepresiahne život hodnotu 100. Pokiaľ áno, život ostane na hodnote, na ktorej bol pôvodne.



(a) (b)

Obrázek 6.61: Keď duch útočí, hráčovi sa znižuje život. Po použití lekárničky z inventára sa jeho život navýši a lekárnička z inventára zmizne (obrázok b)

Kapitola 7

Meranie

Cieľom merania bolo zistiť, či je hra hrateľná a či v hráčovi vzbudí aspoň trochu strach. Výsledné testovanie prebehlo na piatich užívateľoch rôznych vekových kategórií. Každý z nich mal možnosť si finálnu verziu hry zahrať. Po tom, čo hru odohrali, mali odpovedať na osem otázok. Otázky boli kladené tak, aby zhodnotili výslednú kvalitu hru a prípadne našli chyby, ktoré je potrebné zlepšiť.

Otázky zneli nasledovne:

- Bavila vás hra?
- Ako na vás pôsobilo prostredia?
- Desili vás náhodné zvuky?
- Bol pre vás jumpscare nečakaný?
- Bolo pre vás ťažké vyhnúť sa nepriateľom?
- Podarilo sa vám zozbierať všetky kľúče a dostať sa z labyrintu von?
- Podarilo sa vám hru odohrať na prvý krát?
- Čo by ste na hre zlepšili?

Ich výsledné odpovede ukázali, že hra je zábavná. Prostredie pôsobilo dostatočne temne a strašidelne. Prispeli aj k tomu nepríjemné zvuky, ktoré sa náhodne vyskytovali po labyrinte. Ukázalo sa však, že keď je náhodných zvukov v hre až moc, časom nie sú až tak desivé ako po prvých minútach hrania. Niektorých respondentov labyrint zmiatol a ostali stratení a zmätení, nevedeli kde sa vlastne nachádzajú. Čo sa týka nepriateľov, mali respondenti najväčší problém práve s duchom, ktorý sa zjavil vždy nečakane, keď neboli pripravení. Dvaja respondenti boli dokonca duchom zabití a museli hru hrať odznovu. Hru sa podarilo dokončiť všetkým. Avšak jednému užívateľovi musela byť hra spustená odznovu, lebo sa ukázalo, že nie všetky kľúče sa umiestňujú správne a prepadávajú preč. Užívatelia by do budúcnosti uvítali lepšie rozloženie predmetov v miestnostiach, lebo občas sa vygenerovali na takých miestach, kde bolo pre nich obtiažne prejsť. Vďaka testovaniu, sa podarilo odhaliť aj chyby. ktoré je v budúcnosti potrebné ešte doladiť (hra sa občas nespustila na prvý krát, labyrintu chýbala nejaká stena, pri nahrádzaní steny dverami nebola stena zmazaná a nedalo sa prejsť). Respondenti by určite uvítali vylepšenie hry prípadne jej rozšírenia v budúcnosti ako napríklad pridanie viacerých poschodí, prípadne viac jumpscarov.

Kapitola 8

Záver

Cieľom tejto práce bolo vytvoriť herné demo, ktoré bude obsahovať čo najviac procedurálne generovaných prvkov a prvkov, ktoré budú v hráčovi vyvolávať strach.

Pred implementáciou hry bolo potrebné preskúmať herný trh a zistiť, v čom majú dnešné hororové hry nedostatky. Okrem toho bolo potrebné si naštudovať teóriu o strachu a zistiť, aké rôzne fóbie a strachy ľudia v dnešnej dobe majú. Pred samotnou implementáciou bolo nutné, si vybrať, ktorý herný engine bude najvhodnejší a potom si naštudovať jeho funkcionalitu. Ide o autorovu prvú prácu s enginom Unity.

Po štúdiu všetkých potrebných informácií mohlo dôjsť k samotnému návrhu, ktorý bol potom implementovaný.

Výsledné demo obsahuje procedurálne generovaný labyrint, ktorý je po každom spustení hry inak rozložený. V labyrinte sú náhodne rozmiestnené rôzne objekty a nástrahy, ktoré na hráča v priebehu hrania číhajú. Aby hra nebola taká jednoduchá, boli do nej pridaní aj nepriatelia, ktorý usilujú o hráčov život.

Keď bola hra naimplementovaná, mohlo prebehnúť testovanie na užívateľoch. Cieľom bolo zistiť, či je hra hrateľná a či v užívateľoch vyvoláva strach. Ohlasy boli vo veľkej miere kladné a užívateľom sa výsledná hra páčila.

Výsledná aplikácia spĺňa cieľ práce, avšak stále existuje množstvo prvkov, ktoré je možné do hry pridať a tým ju rozšíriť. Jedným z nich je napríklad hra tieňov, ktorá bola spomenutá v návrhu. Labyrint by bolo možné rozšíriť o viac poschodí. Jednotlivé izby by nemuseli byť len štvorcového a obdĺžnikového tvaru, ale mohli by mať rôzne nepravidelné tvary. Hráč by mohol byť schopný bojovať s nepriateľmi a nie sa im len vyhýbať. Napokon by bolo prospešné, vytvoriť vlastné assety a textúry, čím by sa hra stala viac jedinečnou a originálnou.

Literatura

[1] *Unity User Manual* [online]. Dostupné z: https://docs.unity3d.com/Manual/index.html.

[2] Eric Zimmerman, K. S. T. a. *Rules of Play*. First. The MIT Press, 2003. ISBN 0262240459.

[3] Gregory, J. *Game Engine Architecture*. Third. CRC Press, 2018. ISBN 9781138035454.

[4] Hanovská, L. *Člověk a strach: strach v antropologických perspektivách*. Prvný. Togga, 2013. ISBN 978-80-7476-018-1.

[5] Hocking, J. *Unity in Action: Multiplatform Game Development in C#*. Second. Manning, 2018. ISBN 9781617294969.

[6] Jelínek, M. *Strach je přítel vítezú*. Prvný. Starý most, 2011. ISBN 978-80-87338-10-0.

[7] Jim Thompson, B. B.-G. a Cusworth, N. *Game Design: Principles, Practice, and Techniques - The Ultimate Guide for the Aspiring Game Designer*. First. Wiley, 2007. ISBN 978-0471968948.

[8] Lopes, P., Liapis, A. a Yannakakis, G. N. *Sonancia: Sonification of procedurally generated game levels* [online]. 2015. Dostupné z: http://phil-lopes.com/wp-content/uploads/2015/12/ICCC_SonifyingLevels_Final_v2.pdf.

[9] Lopes, P., Liapis, A. a Yannakakis, G. N. *Targeting horror via level and soundscape generation* [online]. 2015. Dostupné z: https://antoniosliapis.com/papers/targeting_horror_via_level_and_soundscape_generation.pdf.

[10] Paris, B.-A. *Unity Game Development Cookbook: Essentials for Every Game*. Buttfield-Addison Paris, 2019. ISBN 9781491999158.

[11] Perron, B. a Barker, C. *Horror Video Games: Essays on the Fusion of Fear and Play*. McFarland & Co; Illustrated edition, 2009. ISBN 978-0786441976.

[12] ReuterskiÖld, L. *Fears, anxieties and cognitive- behavioral treatment of specific phobias in youth* [online]. 2009. Dostupné z: http://www.diva-portal.org/smash/get/diva2:200180/FULLTEXT01.pdf.

[13] Saunders, K. a Novak, J. *Game Development Essentials: Game Interface Design*. Second. Cengage Learning, 2012. ISBN 978-1111642884.

[14] Schell, J. *The Art of Game Design* [online]. Second. Charles River Media, 2014. ISBN 9780429169755. Dostupné z: https://www.inventoridigiochi.it/wp-content/uploads/2020/07/art-of-game-design.pdf.

[15] Short, T. a Adams, T. *Procedural Generation in Game Design*. First. A K Peters/CRC Press, 2017. ISBN 1498799191.

[16] Wolf, D. *Jak překonat strach, úzkost, paniku, fobie*. Prvný. Grada, 2018. ISBN 978-80-271-0618-9.