Ekonomická fakulta
Faculty of Economics
Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované ekonomie a ekonomiky

Bakalářská práce

Vývoj aplikace pro simulaci hospodářského cyklu: uživatelské prostředí

Autor práce: Pavel Kocian
Vedoucí práce: Mgr. Radim Remeš, Ph.D.

České Budějovice
2023

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Pavel KOCIAN**
Osobní číslo:	**E20056**
Studijní program:	**B6209 Systémové inženýrství a informatika**
Studijní obor:	**Ekonomická informatika**
Téma práce:	**Vývoj aplikace pro simulaci hospodářského cyklu: uživatelské prostředí**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování

Cílem práce je vytvořit uživatelskou část 2D počítačové hry, ve které se pomocí použití hospodářské politiky ovlivňuje průběh hospodářského cyklu. Vývoj bude probíhat v herním enginu Unity a v programovacím jazyku C#.

Metodický postup:

1. Studium odborné literatury.
2. Popis použitých technologií pro vývoj aplikace.
3. Návrh, popis vývoje a implementace aplikace.
4. Zhodnocení použitelnosti aplikace pro nasazení v reálném prostředí.
5. Závěr a doporučení.

Rozsah pracovní zprávy:	**40 – 50 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. Halpern, J. (2019). *Developing 2D Games with Unity.* Apress. Dostupné z: <https://link.springer.com/book/10.1007/978-1-4842-3772-4>
2. Blackman, S., & Tuliper, A. (2016). *Learn Unity for Windows 10 Game Development.* Apress. Dostupné z: <https://link.springer.com/book/10.1007/978-1-4302-6757-7>
3. Thorn, A. (2014). *Pro Unity Game Development with C#.* Apress. Dostupné z: <https://link.springer.com/book/10.1007/978-1-4302-6745-4>
4. Thorn, A. (2013). *Learn Unity For 2D Game Development.* Apress. Dostupné z: <https://link.springer.com/book/10.1007/978-1-4302-6230-5>

Vedoucí bakalářské práce: **Mgr. Radim Remeš, Ph.D.**
Katedra aplikované matematiky a informatiky

Datum zadání bakalářské práce: **11. ledna 2022**
Termín odevzdání bakalářské práce: **14. dubna 2023**

doc. Dr. Ing. Dagmar Škodová Parmová
děkanka

doc. RNDr. Tomáš Mrkvička, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 16. března 2022

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne …… Pavel Kocian ……

Poděkování

Mé poděkování patří panu Mgr. Radimu Remešovi, Ph.D., za odborné vedení, vstřícnost a cenné připomínky při zpracování této bakalářské práce.

Obsah

1 Úvod

Cílem této práce je vytvoření uživatelského prostředí pro 2D vzdělávací hru, ve kterém se uživatel může pohybovat a přecházet mezi institucemi, u kterých může uplatňovat faktory hospodářské politiky. Tato hra by měla sloužit jako pomocný nástroj při vzdělávání studentů středních a vysokých škol o hospodářské politice. Je proto důležité, aby uživatelské prostředí bylo intuitivní a působilo na studenta příjemným dojmem, který by jej měl motivovat ke znovuspuštění hry.

V rešeršní části této práce je diskutována základní problematika a terminologie vývoje uživatelského prostředí 2D počítačové hry. Jsou zde přiblíženy základní typy počítačové grafiky a některé nástroje usnadňující její tvorbu. Obdobně jsou popsány možné způsoby tvorby hudby a zvukových efektů, včetně nejčastěji využívaných nástrojů. Dále je v této části práce přiblížena problematika herních enginů využívaných při tvorbě počítačových her.

Praktická část práce popisuje proces vývoje hry, zahrnující tvorbu funkčních mechanik menu, hlavní postavy, mraků, nabídek institucí, generování grafu z dat, hodin, deště a úvodních instrukcí. Tato část také pojednává o tvorbě grafických a zvukových prvků hry.

2 Počítačové hry

Hry jsou v očích mnoha lidí považovány za pouhý zdroj zábavy a náhradu spíše tradičních volnočasových aktivit. Řada výzkumů, ale ukazuje, že hry mohou být velice účinným nástrojem ve vzdělávání, a to ať už zvýšeným zapojením studentů v dané aktivitě nebo posílením schopnosti se učit díky interaktivnímu prostředí (Clark et al., 2016).

Dle Gee (2004) mají studenti, kteří často hrají hry, větší pravděpodobnost, že díky této formě vzdělávání se budou spíše aktivněji zapojovat do celého procesu učení. Hry mohou také podporovat aktivní formu učení poskytováním příležitostí k prozkoumávání, experimentování a objevování nových konceptů v bezrizikovém prostředí beze strachu zeselhání nebo jiných negativních důsledků. Kromě toho jsou také schopny poskytnout individuálně přizpůsobenou okamžitou zpětnou vazbu, která může studentům pomoci identifikovat oblasti, ve kterých musí odpovídajícím způsobem zlepšit své znalosti nebo upravit své vzdělávací postupy (Gee, 2004).

Mnoho her není navrženo čistě se záměrem vzdělávání studentů, ale aby například rozvíjely kritické myšlení, řešení problémů a rozhodovací schopnosti, které jsou nezbytné pro úspěch ve škole i mimo ni. Logické hry mohou studentům pomoci rozvíjet logické myšlení a prostorové uvažování, zatímco strategické hry mohou, kromě strategického myšlení, pomoci rozvíjet například i myšlení analytické. Lze je také použít v různých vzdělávacích prostředích od tradičních učeben po online prostředí. Vzdělávací hry mohou pokrýt širokou škálu předmětů a témat, kdy obvykle záleží jen na vývojářích, o čem by hra měla být (Clark et al., 2016).

2.1 Vzdělávací hry

Mezi hry, které lze zařadit do skupiny vzdělávacích, patří i velice oblíbený Minecraft. Tato hra umožňuje vytvářet své vlastní virtuální světy, ve kterých poskytuje řadu příležitostí k rozvoji kreativity, řešení problémů a týmové spolupráce („About Minecraft", 2023). Další velice oblíbenou hrou je kvízová platforma Kahoot, která umožňuje vytvářet a sdílet kvízy, které poskytují interaktivní způsob ověřování znalostí studentů nebo mohou být vhodným nástrojem i pro představování nových konceptů („What Is Kahoot!?", 2023).

Výbornou ukázkou dobře vytvořené vzdělávací hry je The Oregon Trail (Oregonská stezka), jež byla vytvořena v roce 1990 pro platformu MS-DOS. Původně edukativní software pro americké školáky si díky svému poutavému příběhu získal velkou oblibu a v roce 2019 se z něj stala dedikovaná kapesní hra (Konfršt, 2019).

Vzdělávací hry v oblasti ekonomie pomáhají studentům zkoumat, chápat a uplatňovat ekonomické principy tím, že simulují situace v reálném světě. Mohou být navrženy tak, aby pokryly širokou škálu témat včetně mikroekonomie, makroekonomie, mezinárodního obchodu a behaviorální ekonomie (Bellotti et al., 2012). Aktuálně existuje celá řada her, u kterých studenti mohou rozvíjet své ekonomické znalosti. Jedna z nejznámějších her, Monopoly, hráče seznamuje se základními ekonomickými pojmy, jako je vlastnictví majetku, konkurence na trhu či finanční řízení. Přestože hra Monopoly nebyla vysloveně navržena pro vzdělávací účely, poskytuje studentům cennou příležitost si vyzkoušet ekonomické principy v zábavném kontextu (Gad, 2022).

Mezi digitální ekonomické hry, které byly vytvořeny za účelem vzdělávání, patří řada významných titulů. Hru The Trading Game (Obchodní hra) vytvořil vývojář Toph Tucker s cílem přiblížit lidem akciový trh. Každý hráč má možnost akcie nakupovat a prodávat dle své libosti, přičemž po uzavření třicetisekundové simulace vývoje jedné akcie má možnost porovnat svůj výsledek s ostatními hráči nebo i s celkovou výnosností akcie jako takové. Hra vychází ze skutečného pohybu akcií určitých společností v minulosti. (Tucker & Pearce, 2015).

Online platforma economics-games.com dokonce nabízí celkem čtrnáct her a simulací navržených speciálně pro výuku a studium ekonomie. Web, který vytvořil Nicolas Gruyer s technickou pomocí vývojáře Nicolase Toublanca, si klade za cíl poskytnout pedagogům a studentům interaktivní nástroje, které pomohou vysvětlit ekonomické koncepty zábavným a intuitivním způsobem. Platforma je navržena tak, aby ji bylo možné snadno integrovat do prostředí třídy díky tomu, že je přístupná prostřednictvím libovolného prohlížeče a umožňuje pedagogům i studentům přistupovat ke hrám a simulacím odkudkoliv s připojením k internetu (Gruyer & Toublanc, b.r.).

Hra, která se svým principem nejvíce podobá verzi zpracované v této práci, je Econland z roku 2018, jíž vytvořil dr. Tim Rogmans. Jedná se o makroekonomickou simulační hru, kdy hráči mohou v časových obdobích upravovat úrokovou sazbu, sazbu daně z příjmu právnických a fyzických osob a státní výdaje. Přitom jsou hodnoceni na základně výsledku růstu hrubého domácího produktu, míry nezaměstnanosti, míry inflace. Výsledky jsou prezentovány pomocí grafů a tabulek, na jejichž základně mohou hráči upravovat již zmíněné hodnoty. Hra ale neobsahuje žádné jiné interaktivní uživatelské prostředí, stejně jako většina ostatních her se zaměřením na výuku ekonomie (Rogmans, 2018). Ve své práci, která analyzuje vliv využití hry pro studijní účely, uvádí, že studenti, u kterých byla hra Econland součástí výuky, měli v průměru o jedenáct procentních bodů lepší výsledky nežli studenti, kteří hru nevyzkoušeli. Zároveň na základě dotazníkového šetření studenti, u kterých byla hra součástí výuky, uváděli o 16 % vyšší úroveň zapojení v hodině, nežli studenti v identické hodině bez využití hry Econland (Rogmans, 2022).

3 Herní komponenty

Herní komponenty neboli assety odkazují na vizuální a zvukové komponenty hry, jakými jsou například postavy, pozadí, zvukové efekty a hudba. Jedná se tedy o základní kameny každé hry, které mají za úkol vytvořit vhodnou atmosféru a poutavé prostředí. Díky své důležitosti se ve velkých herních studiích práce na jednotlivých typech assetů pevně odděluje do jednotlivých týmů, dle jejich specializace. Toto se však nemusí vztahovat na menší nezávislá studia, která nemají dostatečný rozpočet. I v těchto studiích bývají většinou alespoň dva hlavní týmy, kdy se jeden z nich zaměřuje na tvorbu grafických a druhý na tvorbu zvukových assetů (Saunders & Novak, 2012).

3.1 Sprite

Při vývoji 2D her se využívá několik typů assetů. Za sprity se označují 2D a 3D obrázky, které mohou představovat postavy nebo předměty ve hře. Obvykle jsou uloženy ve sprite sheetu, což je jeden obrázkový soubor, který obsahuje celou řadu obrázků, jež mohou tvořit celkovou animaci postavy. Sprite sheety dovolují hernímu enginu snadno zobrazovat správný snímek z animace ve správnou chvíli, což je stěžejní pro vhodnou vizualizaci při pohybu nebo interakci s ostatními herními objekty (Saunders & Novak, 2012).

3.2 Prefab

Ne vždy je ale vhodné vytvářet komplikovaný objekt jako jeden. Pokud si nejsme jisti, že v případě znovupoužití daného objektu nebude docházet k žádným změnám, tak pro vytváření komplexních herních objektů se často využívá kombinace několika spritů, které se říká prefab. Díky nim mohou vývojáři vytvářet postavy, scény nebo herní rekvizity rychle a snadno, aniž by vždy museli začínat od nuly. Stačí pouze využít řadu již předem vytvořených spritů a jejich kombinací dosáhnout cíle. Díky tomu jsou také vývojáři schopni dosáhnout lepší a časově méně náročné konzistence napříč celou hrou (Unity Technologies, 2018).

3.3 Script

Velice důležitou součástí každé hry jsou assety ve formě scriptů. Script je v podstatě soubor instrukcí napsaných v programovacím jazyce, který hře říká, co má dělat. Zároveň musí být pečlivě integrovány do celkové hry, aby poskytovaly požadovanou funkčnost a efekt. Může se jednat o základní procesy, jakými je například pohyb hráče na základě sady vstupů z klávesnice nebo i o chování umělé inteligence, která může reagovat na události v herním světě (Rollings & Adams, 2003).

3.4 Tileset

Tilesety se často používají ve hrách, které mají své prostředí založeno na mřížkovém rozložení. Takovými hrami jsou například hry plošinové. Zde je herní prostředí rozděleno na mřížku tileů (destiček), z nichž každý lze nahradit jiným tilem z tilesetu. Stejně jako v případě prefabu nemusejí vývojáři v případě tvorby nového prostředí začínat vždy od začátku, což i zde pomáhá s udržením konzistence napříč celou hrou. To je zejména užitečné při vytváření her s otevřených světem a mnoha různými prostředími, jako jsou lesy, pouště, města nebo cesty.

Díky tomu mohou vývojáři snadno vytvářet komplexní a rozmanitá herní prostředí za použití relativně nízkého počtu stavebních bloků, jimiž jsou tilesety. Jejich výhodou je také to, že vývojáři mohou lépe optimalizovat výkon své hry tím, že najednou nemusejí do paměti načítat velkou sadu samostatných assetů (Adams, 2010).

4 2D počítačová grafika

Grafika hraje v herním designu zásadní roli, neboť utváří estetiku, atmosféru a celkový hráčský zážitek. Správná volba grafického stylu pro hru je klíčová pro to, aby správně odpovídala tématu, příběhu, ale také aby vyhovovala preferencím cílového publika (Addo, 2017).

4.1 Vektorová grafika

Vektorová grafika se při definování tvarů spoléhá na matematické vzorce. Pomocí bodů, čar, křivek, mnohoúhelníků a dalších základních a přesně definovaných úvarů jsou skládány komplexní obrázky, které jsou buďto vyplněny, nebo ohraničeny barvou. Díky tomuto je jednou z nejvýznamnějších výhod vektorové grafiky její nezávislost na rozlišení, což umožňuje škálování obrázků bez ztráty kvality. V důsledku jejich matematického popisu bývají obvykle tyto soubory menší velikosti nežli jejich rastrové protějšky. Jestliže však komplexnost vektorového obrázku překročí určitou mez, může být náročnější na operační paměť a výpočetní výkon nežli grafika rastrová (Todd, 2023).

4.2 Rastrová grafika

Rastrová grafika, často označovaná také jako bitmapová, se skládá z jednotlivých pixelů upořádaných do pravoúhlé mřížky, přičemž každému pixelu je individuálně přiřazena specifická hodnota barvy. Díky tomu jde vytvářet vysoce detailní a realistické obrázky, textury a světelné efekty. Výsledná kvalita obrázku je tedy dána rozlišením, což je počet pixelů na jednotku plochy. Nejčastěji bývá využívána jednotka PPI, která udává počet pixelů na jeden palec čtvereční, ale existuje také metrická jednotka PPCM, která udává počet pixelů na jeden centimetr čtvereční (Gregersen, 2022).

4.2.1 Pixel Art

Pixel art je charakteristický využíváním jednotlivých pixelů k vytváření komplexnějších obrázků, kde je oproti většině stylů, které využívají rastrovou grafiku, žádoucí, aby jednotlivé pixely byly snadno rozpoznatelné. Tento grafický styl se objevil jako přímý důsledek hardwarových omezení v době vzniku arkádových her v pozdních sedmdesátých letech. Umožnil vývojářům vytvářet vizuálně zajímavý obsah v rámci omezení své doby, jakými byly například displeje s nízkým rozlišením, omezené barevné palety nebo nedostatek operační paměti. V dnešní době se pixel art stává populárnějším, což je často přisuzováno jeho nostalgickému vzhledu. Pro vývojáře je toto do jisté míry výhodou, neboť díky jeho jednoduchosti mohou ušetřit značnou část produkčních nákladů (Barnes, 2022).

4.3 Nástroje pro digitální tvorbu grafiky

Řada grafiků využívá při tvorbě digitálních ilustrací grafické tablety. Ty se, narozdíl od počítačové myši, snaží napodobit pocit tradičního kreslení na papír. Pomocí stylusu reprezentujícího pero může grafik kreslit přímo na plochu grafického tabletu a jeho výsledek je zobrazen na obrazovce počítače. Některé moderní grafické tablety jsou rozšířeny i o displej, na kterém lze vidět úpravy v reálném čase. To může grafikovi pomoci například v případě, kdy potřebuje přesně obkreslit vzorový objekt.

Díky implementaci displeje do grafického tabletu je odstraněna bariéra prostoru mezi rukou a obrazovkou, což tvoří celý proces intuitivnějším. U některých modelů grafických tabletů jsou také implementovány funkce jako sledování tlaku pera na grafickém tabletu nebo i rozpoznání náklonu a rotace pera. Jestliže jsou tyto funkce podporovány i ze strany softwaru, je proces digitální tvorby ještě realističtější (Williams, 2020).

5 Zvuk

Zvukové efekty jsou používány k poskytování zpětné vazby pro různé herní události, akce uživatele nebo i pro jednotlivé prvky prostředí. Mohou také pomoci naznačit hráčův úspěch či neúspěch nebo jen obecně zlepšit hráčův pocit ze hry. K jeho dobrému pocitu může přispět i správně zvolená podkladová hudba (Holmes, 2022).

5.1 Zaznamenávání zvuku

Pro zaznamenávání zvuku jsou jako primární nástroje používány mikrofony, které převádějí zvukové vlny na elektrické signály, jež lze následně zpracovat a zaznamenat. Nejčastěji se pro nahrávání používají mikrofony dynamické a kondenzátorové, které se liší způsobem převodu zvukových vln na elektrický signál. Za citlivější a přesnější se považují mikrofony kondenzátorové, které ovšem potřebují ke svému správnému fungování takzvané fantomové napájení o elektrickém napětí 48 V. Toto napájení je nejčastěji mikrofonu dodáváno skrze adekvátní zvukové rozhraní, kterým může být například mixpult (Bartlett & Bartlett, 1999).

5.2 Nástroje pro digitální tvorbu zvuku

Ačkoliv řada programů pro tvorbu digitálního zvuku podporuje virtuální klávesy, které převádějí vstup ze standardní klávesnice na adekvátní noty, ne vždy je tento proces intuitivní a pohodlný. Pro usnadnění procesu tvorby digitální hudby existuje celá řada MIDI[1] kontrolerů různých forem, kdy se nejčastěji užívá imitace kláves klavíru. Tyto kontrolery slouží jako vstupní zařízení pro práci s programy určenými k tvorbě digitálních zvuků (Nugent, 2021).

K úpravě nebo tvorbě nových zvukových vzorků jsou využívány syntezátory, které jsou pomocí různých technik pro generování a manipulaci zvuku schopné vytvořit široké spektrum zvuků dříve neslyšených nebo reprezentujících určité hudební nástroje. Syntezátory musejí mít alespoň jeden oscilátor, který je využit k vytvoření úvodního zvuku. Ten je následně ovlivňován celou řadou prvků syntezátoru, jakými mohou být například frekvenční a vlnová modulace (Nugent, 2020).

[1] Musical Instrument Digital Interface (Digitální rozhraní pro hudební nástroj)

6 Herní engine

Herní engine je softwarový framework[2], určený k usnadnění vývoje videoher. Zahrnuje kolekci nástrojů, knihoven a komponent, které se starají o naprosto nezbytné součásti her, jako je vykreslování grafiky, simulace fyziky, správa vstupů a zpracování zvuku. Výběr vhodného herního enginu je klíčovým rozhodnutím pro úspěch celého projektu. To platí i pro 2D hry, kde engine hraje významnou roli v celkové optimalizaci a funkčnosti hry. Pro správné rozhodnutí je třeba zvážit celou řadu faktorů, jejichž důležitost, počet a podrobnost se může značně měnit dle požadavků na finální hru (Martin, 2020).

Obecně lze rozdělit herní enginy na dvě základní kategorie, a to na ty vytvořené třetí stranou a ty vyvinuté přímo herním studiem. Herní enginy vytvořené třetí stranou, která se soustřeďuje především na licencování svého herního enginu ostatním vývojářům místo toho, aby vytvářela hry vlastní, mají své značné výhody především pro nezávislé vývojáře nebo malá herní studia. Jelikož proces vývoje herního enginu je náročný a drahý, mohou vývojáři použitím herního enginu třetí strany tento krok při vývoji her přeskočit a použít již existující nástroj (Martin, 2020).

Ačkoliv se to na první pohled může zdát jako výhoda, neboť většina malých vývojářů nemá prostředky na vývoj vlastního herního enginu, nevýhodou zůstává nutnost platit licenční poplatky. Pokud tedy vývojáři předpokládají prodej velkého množství kopií své hry, často se uchylují k vývoji vlastního herního enginu. Díky tomu nemusejí platit žádné poplatky třetí straně. Navíc jim to dovoluje vytvořit mnohem specifičtější nástroj k dosažení jejich cíle, což může vést k lepšímu finálnímu hernímu zážitku. Například, pokud je cílem vytvořit hru s perfektní imitací lidského obličeje, může herní engine navržený od základu s tímto cílem být mnohem efektivnějším nástrojem než použití herního enginu třetí strany. Ty se totiž většinou soustřeďují na nabídku co nejvíce funkcí, které by mohly využít různí vývojáři. Díky nabídce velkého množství nástrojů, jež mohou vývojáři použít pro tvorbu dvourozměrných nebo trojrozměrných her nebo případnému exportu pro různé herní platformy jsou schopni získat mnohem větší množství vývojářů, kteří si budou licencovat jejich produkt. Při vývoji 2D her jsou momentálně využívány především herní enginy Unity a Godot (Martin, 2020).

[2] Aplikační rámec, vývojová platforma.

6.1 Unity

Unity je multiplatformní herní engine vytvořený společností Unity Technologies v roce 2005, který se stal velice oblíbeným nástrojem pro vývoj her. Vyniká svou flexibilitou a jednoduchostí použití, díky čemuž je ideální volbou jak pro začínající, tak i zkušené herní vývojáře. Jednou z velkých výhod Unity je jejich Asset Store – obchod, který poskytuje nepřeberné množství již vytvořených assetů, které lze snadno integrovat do projektů. Tyto assety lze zakoupit přímo od jejich tvůrců, ale lze zde najít velké množství assetů zdarma dostupných. Dalším významným bonusem Unity je jeho komunita, kterou tvoří různorodá skupina vývojářů, sdílejících své znalosti, zkušenosti a postřehy s ostatními vývojáři. Díky tomu existuje velké množství online fór a návodů, pomocí nichž se lze více soustředit na skutečný vývoj, a nikoliv na pročítání sáhodlouhé dokumentace, která ne vždy dokáže poradit (Sinicki, 2021).

Neocenitelným nástrojem je Unity Visual Scripting, což je oficiální scriptovací nástroj navržený tak, aby umožnil vývojářům a grafickým návrhářům vytvářet herní mechanismy a interakce postav, aniž by se museli spoléhat na programovací jazyk C# využívaný Unity. Scriptování probíhá propojováním jednotlivých logických uzlů, což značně snižuje vstupní bariéru u vývoje her pro skupinu lidí, kteří nemají zkušenost s programováním (Elias, 2021).

Pro tvoření dynamických animací je součástí Unity také Animator Controller (animační kontroler). Každý objekt může mít přiřazenu komponentu Animator, která slouží jako spojení mezi animací a Animator Controllerem. Ta je zodpovědná za ovládání přehrávaných animací, správu jejich stavu a prolínání mezi různými animacemi daného objektu. O tom, kdy a jaká animace se má spustit rozhoduje Animator Conroller na základě vytvořené logiky (Unity Technologies, b.r).

V rámci licencování nabízí Unity několik variant: Personal, Plus, Pro a Enterprise. Tyto varianty jsou od sebe odlišeny funkcemi, které nabízejí, roční cenou za licenci pro vývojáře nebo i maximálním ročním obratem kterého můžete v rámci daného plánu dosáhnout. V tabulce 1 můžeme nalézt zkrácené základní porovnání mezi jednotlivými licenčními plány z informací, které jsou na stránkách Unity uvedeny v porovnávači plánů a v podmínkách užití softwaru (Unity Technologies, b.r.; Unity Technologies, 2022).

Tabulka 1: Základní porovnání mezi licenčními verzemi Unity

Faktor / Licenční verze	**Personal**	**Plus**	**Pro**	**Enterprise**
Cena za jednoho vývojáře na rok	Zdarma	399 USD	2040 USD	Cena je domluvena individuálně
Maximální roční obrat a financování za rok	100 000 USD	200 000 USD	Není omezeno	Není omezeno
Možnost úpravy úvodní obrazovky při spuštění hry	Ne	Ano	Ano	Ano
Tvorba pro konzolové platformy	Ne	Ne	Ano	Ano
Prodloužená dlouhodobá podpora projektu na 3 roky	Ne	Ne	Ne	Ano

Zdroj: Unity Technologies, vytvořeno autorem

Z tohoto základního porovnání lze vyčíst, že varianta Personal je určena především pro nadšence a malé vývojáře. Verze Plus je téměř identická s verzí Personal, ale nabízí jeden velký benefit, kterým je možnost upravit úvodní obrazovku, která se zobrazí vždy při spuštění vytvořené hry. Ve verzi Personal jí vždy bude úvodní animace Unity, která může na hráče hned po spuštění působit neprofesionálním dojmem tím, že je často spojována s hrami na nižší úrovni vývoje (YeeOfficer, 2020).

6.2 Godot

Godot je multiplatformní open-source[3] herní engine, který byl po dlouhou dobu optimalizovaný především pro tvorbu dvojrozměrných her, ačkoliv podporoval i tvorbu her trojrozměrných. To, že je Godot open-source program, bylo po dlouhou dobu dvojsečné. Na jedné straně je velká výhoda to, že vývojáři nemusejí platit žádné licenční poplatky za využívání Godot enginu pro své hry, na straně druhé je pomalý vývoj vylepšení a oprav existujících problémů obecně spjatý s open-source projekty. Mezi hlavní důvody, proč vývojáři preferovali použití jiných herních enginů, patří především neideální zpracování fyziky pro 2D objekty, nekonzistentní terminologie napříč enginem, která může působit zmatečně, i špatná práce s osvětlením objektů (Wirtz, 2019).

Vyřešení řady těchto problémů a spoustu dalších vylepšení přináší nová verze Godot 4, která byla oficiálně vydána 1. března 2023 s podporou více než dvou tisíc vývojářů, kteří se na projektu podíleli. Je tedy otázkou času, jak tato verze bude komunitou přijata, a jestli si Godot engine získá větší oblibu u vývojářů (Godot Engine, 2023).

[3] Software s otevřeným zdrojovým kódem.

7 Vývoj aplikace

Pro tvorbu aplikace byl zvolen herní engine Unity, neboť jeho prostředí je intuitivní pro ovládání, získává průběžné aktualizace chyb a jeho součástí je i rozsáhlá dokumentace, která je dobře strukturalizována, aby jí snadno porozuměl i programátor začátečník (Halpern, 2019).

Při vytvoření nového projektu v Unity je vždy vytvořena nová scéna, jejíž součástí je objekt hlavní kamery, který při spuštění projektu zobrazuje vše, co je v jeho rozsahu. Při práci s herním enginem Unity je třeba věnovat pozornost především čtyřem oknům. Prvním z nich je okno hierarchie, ve kterém lze vidět strukturu všech objektů ve scéně a případně lze tyto objekty individuálně označit pro následující úpravu v okně scény, kde lze vizuálně upravovat jejich pozici a velikost. Po označení objektu lze také v okně inspektoru vidět a upravovat objektu přiřazené komponenty. Jako poslední hlavní okno lze označit okno projektu, ve kterém je možné vidět všechny assety importované k projektu. Jelikož u řady komponent v inspektoru můžeme vkládat assety a herní objekty i způsobem „drag and drop“, je vhodné mít okna projektu a hierarchie otevřená, aby šlo snadněji přenášet tyto prvky a nemusely být vybírány přímo v nabídce dané komponenty.

Jestliže je objektu přiřazena komponenta v podobě scriptu, je možné se na daný objekt ve scriptu odkázat pomocí „gameObject“. Pokud chceme upravovat některou z komponent přiřazených tomuto objektu, můžeme toho docílit pomocí příkazu „gameObject.GetComponent<T>()“, kdy „T“ zde reprezentuje název požadované komponenty. Pokud by tento vytvořený script potřeboval komunikovat s jinými objekty nebo jejich vlastnostmi, jeden ze způsobů, jak toho dosáhnout, je vytvoření veřejné proměnné uvnitř třídy typu „GameObject“. Díky tomu bude možné v inspektoru u tohoto scriptu zvolit požadovaný objekt u dané proměnné. S tímto přístupem přichází nevýhoda, že daný objekt bude veřejný i pro ostatní scripty nebo místa. K dosažení zobrazení neveřejné proměnné pouze v inspektoru můžeme před modifikátor přístupu napsat atribut „[SerializeField]“.

7.1 Menu

Ve hře uživatel narazí na dvě menu. Prvním je hlavní úvodní menu, které je zobrazeno při spuštění hry a herní menu zobrazené při pozastavení hry.

7.1.1 Hlavní menu

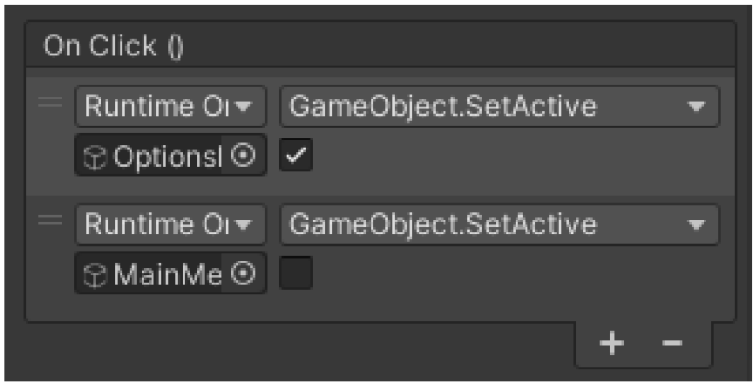
Úvodní hlavní menu je založeno jako samostatná scéna, kde je uživateli prezentována řada tlačítek: spustit hru, výuka, nastavení hry, odejít ze hry a zobrazení informací o hře. Aby bylo možné dosáhnout správné vizualizace těchto tlačítek na pozadí, je třeba v této scéně přidat objekt plátna (Canvas), který reprezentuje abstraktní prostor, ve kterém jsou tlačítka rozmístěna a vykreslována. Zde je třeba vždy u komponenty „Canvas Scaler“ u vlastnosti „UI Scale Mode“ vybrat možnost „Scale With Screen Size“, aby velikost plátna byla zachována napříč různými rozlišeními obrazovek. Všechny prvky uživatelského prostředí, se kterými má uživatel interagovat, musejí být podřazeny tomuto objektu plátna a musejí obsahovat komponentu „Canvas Renderer“, aby je bylo možné vykreslit. V tuto chvíli je možné nastavit velikost a umístění kamery přesně na velikost plátna. Pro zjednodušení lze v komponentě „Canvas“ u plátna nastavit vlastnost „Render mode“ na „Screen Space – Camera“, kdy se po této změně zpřístupní vlastnost přidružení kamery (Render Camera), ve které lze požadovanou kameru zvolit. Tato změna docílí toho, že plátno se automaticky přizpůsobí velikosti a pozici přidružené kamery.

Pro přidání pozadí lze uvnitř objektu plátna vytvořit objekt panelu (Panel) s požadovanou komponentou „Canvas Renderer“ a komponentou „Rect Transform“, která nám dovoluje ukotvit panel ke středu plátna a zároveň změnit jeho velikost tak, aby byla shodná s plátnem. K přidání obrázku pozadí využívá panel komponentu „Image“, kde lze vlastnosti „Source Image“ zvolit sprite, který by se měl na panelu zobrazovat.

Objekty podřazené plátnu jsou také tlačítka, která využívají Unity předdefinované komponenty „Button“, „Rect Transform“, „Canvas Renderer“, „Image“, a mají také přiřazeny předdefinovaný objekt s komponentou „TextMeshPro – Text“ nebo „Image“ dle toho, jestli má tlačítko textový popis nebo ikonu. Komponenta „Button“ dovoluje vývojáři nastavit chování tlačítka, pokud se přes něj přejede kurzorem myši, klikne se na něj, nebo například také rychlost animace při změně barvy tlačítka. Toto může uživateli lépe vizualizovat, že je tlačítko aktivní a lze na něj kliknout.

Abychom mohli vytvořit fungující navigaci v menu a zároveň nemuseli pro každou nabídku tvořit nové plátno nebo deaktivovat prvky individuálně, můžeme požadovanou sadu tlačítek a textů, které se mají zobrazovat společně, obalit prázdným objektem, jemuž budou požadované objekty podřazeny. U tohoto objektu bez přidaných komponent, je možné ovládat zobrazení všech podřazených objektů najednou pomocí metody SetActive() nad seskupujícím objektem, která přejímá parametr typu boolean hodnoty true nebo false, podle toho, jestli chceme, aby byl herní objekt aktivní nebo nikoliv. Tuto metodu můžeme zavolat nad jakýmkoliv objektem ve scéně bez nutnosti psaní scriptu pro spouštějící tlačítko. Jak lze vidět na obrázku 1, komponenta „Button“ nám u tlačítka v inspektoru umožňuje u události OnClick() graficky vybrat objekt nad kterým bude metoda zavolána, metodu nebo událost objektu a v případě vybrání SetActive() i zaškrtávací pole reprezentující hodnoty true a false.

Obrázek 1: Událost OnClick() v komponentě „Button“



Zdroj: Autor (2023)

Obdobně lze vytvořit stejnou funkcionalitu i pomocí scriptu, který přiřadíme tlačítku jako komponentu. V metodě start musíme objektu tlačítka přidat možnost naslouchat k UnityEvent, který je spuštěn po kliknutí na dané tlačítko. V našem případě je možné tohoto dosáhnout pomocí „gameObject.onClick.AddListener()“, kdy metodě „AddListener()“ můžeme jako parametr předat metodu, která by se měla vykonat.

Díky tomu je možné vytvořit komplexní menu, které lze snadno ovládat pomocí klikání tlačítek a přesouvat se ve struktuře menu pouhým aktivováním a deaktivováním požadovaných skupin objektů.

Toto chování nám bohužel nepomůže v případě, kdy tlačítko „Spustit hru“ má spustit samostatnou scénu hry. Pro něj musíme vytvořit novou komponentu v podobě scriptu. Můžeme si tedy vytvořit metodu, jejíž cílem bude přepnout scénu na námi požadovanou. V rámci třídy „SceneManager“ Unity nabízí metodu „LoadScene()“, které lze předat buďto parametr ve formě stringu s názvem požadované scény nebo indexovou hodnotu scény, kterou lze nalézt v nastavení sestavení projektu. Zde je důležité, aby všechny scény, které budou v projektu využity, byly v zaškrtávacím poli označeny za aktivní pro sestavení.

Na obdobný problém lze narazit i u tlačítka pro ukončení hry. Znovu si můžeme vytvořit novou metodu jejíž cílem bude ukončit celý program, čehož dosáhneme pomocí Application.Quit().

V nastavení hlavního menu má uživatel možnost změnit hlasitost hudby v pozadí pomocí jezdce, který lze v Unity vytvořit pomocí předdefinovaného objektu „Slider“, kterému můžeme v komponentě „Slider“, na rozdíl od komponenty „Button“, určit i rozsah nebo výchozí zobrazovanou hodnotu. Dále je zde vytvořeno zaškrtávací pole z předdefinovaného objektu „Toggle“, které může zapnout nebo vypnout úvodní instrukce pro hráče při zapnutí hry.

7.1.2 Herní menu

Velice obdobně jako hlavní menu funguje i menu uvnitř scény hry. Znovu se jedná o sadu tlačítek a textů podřazených seskupujícímu objektu, přičemž zobrazená tlačítka jsou: pokračovat, nastavení, zpět do úvodního hlavního menu a zobrazení informací o hře. Toto herní menu ale není ukázáno vždy a je možné se do něj přepnout pouze pomocí zmáčknutí klávesy „escape“. Stejně jako u hlavního menu je třeba mít objekt plátna, který, ačkoliv nic nezobrazuje, je stále aktivní, aby u něj fungoval nově vytvořený přiřazený script, který ovládá celé fungování menu včetně pozastavení hry.

Jak lze vidět ve výpisu kódu 1, k dosažení reakce na stisk klávesy je zapotřebí kontrolovat, jestli byla klávesa „escape“ stisknuta v metodě „Update()“, která je volána při vygenerování každého nového snímku ve hře. Aby bylo možné pomocí stejné klávesy herní menu i deaktivovat, je třeba vytvořit proměnnou, v tomto případě „isGamePaused“ typu boolean, která kontroluje, jestli je herní menu aktivní, nebo nikoliv. Pokud zmáčkneme klávesu „escape“ a proměnná udržuje hodnotu false, zavolá se námi

vytvořená metoda „Pause()“ ve které se pomocí „SetActive(true)“ aktivuje objekt úvodního herního menu s požadovanými tlačítky a pozadím. Pro dosažení efektu zastavení času pro objekty reagující na herní čas můžeme použít příkaz „Time.timeScale = 0f“. V neposlední řadě je důležité přenastavit proměnnou „isGamePaused“ na hodnotu true, aby v případě dalšího zmačknutí klávesy „escape“ byla hra spuštěna pomocí metody „Resume()“.

Výpis kódu 1: Metoda Update() u scriptu PauseMenu

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (isGamePaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
}
```

Zdroj: Autor (2023)

Metoda „Resume()“ má za cíl uvést vše do původního stavu. Změní tedy „Time.timeScale“ na výchozí hodnotu 1f, proměnnou „isGamePaused“ na hodnotu false a zároveň projde polem všech objektů, které se mají deaktivovat při spuštění. Toto pole herních objektů je naplněno v inspektoru.

V úvodním herním menu má uživatel také tlačítko pro přechod zpátky do úvodního hlavního menu. Aby uživatel nepřišel o veškerý postup ve hře v případě, že na něj klikl omylem, zobrazí se mu nová sada tlačítek žádajících o potvrzení. Jestliže nepotvrdí návrat do úvodního hlavního menu, bude vrácen do úvodního herního menu. V případě potvrzení návratu toto tlačítko reaguje na metodu fungující obdobně jako tlačítka „Spustit hru“ a „Výuka“ v úvodním hlavním menu. I zde je tedy třeba pro „SceneManager.LoadScene()“ definovat parametr názvu nebo indexu úvodního

hlavního menu. V případě, kdyby uživatel hru z úvodního hlavního menu znovu spustil, hodnoty „Time.timescale“ a „isGamePaused“ by stále měly hodnoty nastavené metodou „Pause()“. Abychom tomuto problému předešli, jedním z možných způsobů je zavolání metody „Resume()“ po stisknutí tohoto tlačítka. Tlačítko „Pokračovat“ při jeho stisknutí pouze zavolá metodu „Resume()“ a tlačítko nastavení zobrazí stejné možnosti nastavení jako u úvodního hlavního menu.

7.2 Hlavní charakter

Objekt hlavního charakteru je vytvořen jako prázdný objekt ve scéně, kterému je automaticky přiřazena komponenta „Transform“, která nám dovoluje objekt umístit na scéně. Aby mohl náš objekt mít i vizuální stránku, je třeba přidat komponentu „Sprite Renderer“ a u její vlastnosti „Sprite“ definovat požadovaný sprite, který má objekt mít. V tuto chvíli je hlavní charakter pouze obrázkem, na který nepůsobí žádná fyzika a v případě spuštění hry by se pouze staticky nacházel na přiřazených souřadnicích. To můžeme změnit přidáním komponenty „Rigidbody 2D“, která zapříčiní působení fyziky na náš objekt. V této komponentě můžeme objektu nastavit jeho hmotnost, sílu tření i velikost gravitační síly. Při spuštění hry v tomto stavu na objekt sice působí fyzika, ale neboť objekt nemá nastavené žádné hranice, propadl by skrze jakýkoliv jiný objekt. Je tedy nutné přidat komponentu „Box Collider 2D“ která dovoluje tvorbu kolize mezi dvěma objekty. Pro správné nastavení interakce hlavního charakteru s ostatními objekty můžeme využít vlastnost této komponenty „Edit Collider“, která dovoluje upravit velikost a pozici kolizního boxu kolem našeho objektu. Zároveň nechceme, aby se charakter nakláněl do stran v případě doteku s jiným objektem. K zakázání této akce slouží vlastnost „Constraints“, ve které můžeme objektu zakázat jeho rotaci po ose Z.

Nyní můžeme vytvořit neviditelný objekt, reprezentující podlahu, po kterém se může náš hlavní charakter pohybovat. K tomu nám stačí v hierarchii vytvořit nový prázdný objekt, který pomocí komponenty „Transform“ můžeme umístit na požadované místo a zvětšit jeho velikost natolik, aby svou šířkou pokrýval celý prostor, po kterém může uživatel s charakterem chodit. Na rozdíl od hlavního charakteru nemusí mít tento objekt komponentu „Rigidbody 2D“, neboť nám bude stačit, když bude staticky v prostoru na jednom místě. Ale aby tento objekt byl schopen kolidovat s hlavním charakterem, musíme mu přidat komponentu „Box Collider 2D“, jinak by skrze něj hlavní charakter propadl.

Jelikož v této hře nemá uživatel nekonečný prostor pro pohyb hlavním charakterem do stran, je důležité vytvořit objekty podobné naší podlaze, které budou limitovat hráčův pohyb jako zdi po stranách. Bez těchto limitů by mohl hráč spadnout z herního světa. Vytvořené objekty, s komponentou „Box Collider 2D“ umístíme na konce naší podlahy s adekvátními rozměry, aby je hráč nemohl přeskočit. V tuto chvíli bychom narazili na problém v případě, kdyby se hlavní charakter dotkl těchto objektů ve skoku a snažil se zůstat v pohybu ve směru této zdi. Charakter by zůstal ve vzduchu do doby, než by hráč ukončil svůj pohyb. Tomu můžeme předejít vytvořením materiálu „Physics Material 2D“, kde nastavíme vlastnost tření (Friction) na hodnotu 0. Tento nově vytvořený materiál můžeme předat komponentě „Box Collider 2D“ k vlastnosti „Material“ u zdí, což zapříčiní, že hráč vždy správně spadne na podlahu za jakýchkoliv okolností.

7.2.1 Pohyb postavy

Pro pohyb našeho objektu je třeba vytvořit novou komponentu ve formě scriptu. Protože nechceme, aby hráč měl možnost se pohybovat v případě, že je hra pozastavena nebo se uživatel nachází v herním menu, můžeme ke kontrole, jak lze vidět ve výčtu kódu 2, využít proměnnou „isGamePaused“ objektu herního menu, ve kterém si tuto informaci uchováváme. Jedním ze způsobů, jak lze získávat informaci o tom, že se uživatel chce posunout, je stejně jako u vstupu do herního menu „Input.GetKeyDown()“ s předáním parametru požadované klávesy. Unity ale nabízí mnohem elegantnější způsob, jak tuto informaci získat, a to pomocí „Input.GetAxisRaw()“, kde jako parametr předáváme požadovanou osu ve formátu string. Tímto přístupem můžeme v nastavení projektu v kategorii „Input Manager“ specifikovat hlavní a alternativní klávesy pohybu do stran. Tato metoda vrací tři hodnoty: -1 pro pohyb doleva, 0 pokud pohyb není a 1 pro pohyb doprava. Tuto hodnotu si uložíme do proměnné „dirX“ pro další použití. Alternativně lze použít „Input.GetAxis()“, která může vrátit rozsah od -1 do 1. To způsobuje, že pokud hráč stiskne klávesu pro pohyb, tak na základě senzitivity vstupu, která lze nastavit v „Input Manager“, chvíli trvá, než bude vrácena hodnota 0, a vytváří tím dojem klouzání postavy.

K nastavení pohybu objektu je třeba využít komponentu „Rigidbody 2D“, který je ve výpisu kódu 2 zastoupen proměnnou „rb“, kde chceme změnit pouze rychlost (Velocity) objektu pomocí přiřazení nového dvojrozměrného vektoru. Za první parametr X dosazujeme proměnnou rychlosti, která je založena s atributem „[SerializeField]“ aby

ji bylo možné za běhu programu ručně měnit v inspektoru, násobenou proměnnou „dirX“ která v sobě uchovává informaci o směru pohybu. Druhým parametrem necháváme aktuální pozici Y objektu.

Výpis kódu 2: Pohyb hlavního charakteru

```
void Update()
{
    if (!PauseMenu.isGamePaused)
    {
        dirX = Input.GetAxisRaw("Horizontal");

        rb.velocity = new Vector2(speed * dirX, rb.velocity.y);

        if (Input.GetButtonDown("Jump") && IsGrounded())
        {
            rb.velocity = new Vector2(0, jump);
        }

        UpdateAnimation();
    }
}
```

Zdroj: Autor (2023)

Pro skok potřebujeme kontrolovat, jestli uživatel zmáčkl příslušnou klávesu, kdy pomocí „Input.GetButtonDown() a parametru názvu příslušného tlačítka, je možné znovu v „Input Manager“ definovat všechny klávesy, které pod funkčnost tohoto tlačítka spadají. Problém nastává v okamžiku, kdy by uživatel zmáčkl příslušnou klávesu několikrát v řadě za sebou. Hra by uživateli dovolila vyskočit neomezeně vysoko. K eliminaci tohoto problému je třeba vytvořit metodu, v tomto případě nazvanou „IsGrounded()“, která vrací hodnotu true nebo false. K určení, jestli se objekt skutečně dotýká země a má mu být dovoleno vyskočit znovu, je používáno „Physics2D.BoxCast()“. Tato metoda vytváří neviditelný box, který kontroluje, jestli se nedotýká specifikované vrstvy. Jak lze vidět ve výpisu kódu 3, v našem případě je této metodě předáváno šest parametrů v tomto pořadí: bod ze kterého je box generován, velikost boxu, stupeň otočení boxu, vektor reprezentující směr generování boxu, vzdálenost generovaného boxu od bodu generování a vrstva, se kterou má probíhat

kontrola kolize. Proměnná „bc“ zde reprezentuje komponentu „Box Collider 2D“ a „jumpGround“ typu „LayerMask“ označuje vrstvu do které jsou přiděleny všechny objekty od kterých se smí hráč odrazit, což je třeba u každého takového objektu udělat individuálně.

Výpis kódu 3: Vnitřek metody IsGrounded()

```
return Physics2D.BoxCast(bc.bounds.center, bc.bounds.size, 0f, Vector2.down, .1f, jumpGround);
```

Zdroj: Autor (2023)

Jestliže je tedy splněna podmínka pro skok, tak stejně jako u pohybu je vytvořen nový vektor, u kterého je pro první parametr X předávána hodnota 0 a pro druhý parametr Y je předávána proměnná „jump“, která je založena jako „[SerializeField]“ ze stejného důvodu jako u pohybu proměnná „speed“.

Abychom docílili realističtějšího chování hlavního charakteru, je na závěr volána vytvořená metoda „UpdateAnimation()“. Jak lze vidět ve výpisu kódu 4, tato metoda kontroluje proměnnou „dirX“, ve které je uložena hodnota směru pohybu, na jejímž základě je rozhodnuto, jestli bude animace chůze spuštěna, a případně jestli bude celý charakter obrácen, aby se dosáhlo iluze, že postava chodí vpřed správným směrem. V „Animator Controller“ je třeba vytvořit proměnnou typu boolean napojenou na požadovanou animaci, jež je označená ve výpisu kódu 4 jako „running“, která ovládá průběh animace. Ve scriptu je možné přistoupit k proměnné „anim“ reprezentující komponentu „Animator“. Pomocí metody „SetBool()“ můžeme změnit hodnotu zvolené proměnné, kterou předáváme metodě v podobě typu string jako první parametr. Druhým parametrem je hodnota, na kterou se má zvolená proměnná změnit.

Výpis kódu 4: Metoda UpdateAnimation()

```
private void UpdateAnimation()
{
    if (dirX > 0f)
    {
        anim.SetBool("running", true);
        sprite.flipX = false;
    }
    else if (dirX < 0f)
    {
        anim.SetBool("running", true);
        sprite.flipX = true;
    }
    else
    {
        anim.SetBool("running", false);
    }
}
```

Zdroj: Autor (2023)

Pro obrácení spritu objektu ve směru pohybu lze využít proměnné „sprite“, reprezentující komponentu „Sprite Renderer“, u které můžeme nastavit hodnotu „flipX“ na true nebo false.

7.3 Objekty mraků

Mraky jsou tvořeny čtyřmi různými sprity, ze kterých je tvořen prefab, jenž obsahuje script pro jejich individuální pohyb. Aby nebylo nutné mít všechny mraky rozmístěny na scéně se specifickými hodnotami velikosti, pozice a rychlosti pohybu, můžeme vytvořit generátor těchto prefabů, který bude tvořit nové kopie mraků.

K vytvoření tohoto generátoru je třeba vytvořit nový prázdný objekt, který umístíme na požadovanou pozici, ze které by se měly mraky generovat. Tomuto objektu vytvoříme novou komponentu ve formě scriptu. Je zde vytvořeno pole herních objektů, ve kterém jsou uloženy jednotlivé prefaby mraků ke generaci. I zde je důležité nejprve zkontrolovat proměnnou „isGamePaused“, aby nedocházelo ke generaci mraků při pozastavení hry. Tato proměnná je kontrolována v samostatné metodě „AttempSpawn()“, která vždy vyvolává sama sebe pomocí metody „Invoke()“, kdy prvním parametrem je název volané metody ve formátu string a druhým parametrem je doba, po jejímž uplynutí má být

požadovaná metoda vyvolána. Pokud hra pozastavena není, zavolá metodu „SpawnCloud()“ s parametrem pozice objektu generátoru, který je v metodě „Start()“ uložen do proměnné „startPos“ pomocí „transform.position“.

Jak lze vidět ve výpisu kódu 5, pokud je metoda „SpawnCloud()“ zavolána, vytvoří instanci pomocí metody „Instantiate()“, nového herního objektu mraku (cloud) z pole mraků (clouds). Výběr inicializovaného objektu je náhodný, aby se dosáhlo větší rozmanitosti při generaci a obloha nevypadala příliš stroze. Náhodného výběru je dosaženo pomocí metody „Random.Range()“, u které je prvním argumentem minimum rozsahu a druhým maximum. Ze stejného důvodu rozmanitosti je pro nový objekt vybrána i náhodná Y pozice, kdy proměnná „hightDifference“ určuje maximální odchylku objektu od pozice objektu generátoru. Stejně tak je pomocí proměnných „minSize“ a „maxSize“ náhodně zvolen rozsah velikosti mraku.

Výpis kódu 5: Metoda SpawnCloud()

```
void SpawnCloud(Vector2 startPos)
{
    GameObject cloud = Instantiate(clouds[Random.Range(0, clouds.Length)]);
    float startY = Random.Range(startPos.y - hightDifference, startPos.y + hightDifference);
    cloud.transform.position = new Vector2(startPos.x, startY);
    float scale = Random.Range(minSize, maxSize);
    cloud.transform.localScale = new Vector2(scale, scale);
    cloud.GetComponent<CloudScript>().StartFloating(
        Random.Range(minSpeed, maxSpeed), endPoint.transform.position.x);
}
```

Zdroj: Autor (2023)

Tím, že objekt inicializovaného mraku je prefab obsahující script pro pohyb (CloudScript), můžeme k této komponentě přistoupit a zavolat metodu „StartFloating()“, které se předávají dva parametry. Prvním je rychlost pohybu mraku z náhodného rozsahu a druhým je souřadnice X, u které by měl být objekt zničen. V tomto případě byl vytvořen prázdný objekt, jenž lze ve scéně snadno umístit a metodě je předávána jeho pozice X. Výhodou tohoto přístupu, oproti přímému určení souřadnice, je možnost vidět skutečné umístění daného objektu ve scéně, a tedy přesné pozice u které budou mraky zničeny.

Metoda „StartFloating()“ slouží uvnitř scriptu „CloudScript“ pro předání a uložení informací o jejích parametrech do proměnných. Abychom nemuseli používat cyklus,

který by pohyboval mraky, můžeme využít metodu „Update()“, ve které využijeme uložené hodnoty získané z metody „StartFloating()“.

Aby se mraky nepohybovaly v případě, že je hra pozastavena, i zde se kontroluje proměnná „isGamePaused“, jak lze vidět ve výpisu kódu 6. Jestliže hra zastavena není a chceme aby se mraky pohybovaly, můžeme toho docílit zavoláním metody „Translate()“ nad komponentou „Transform“ zastoupenou proměnnou „transform“. Této metodě předáváme parametr ve formě vektoru, který udává směr a vzdálenost posunu. Zkrácený zápis „Vector2.right“ je ekvivalentní k zápisu „Vector2(1,0)“ a udává, že objekt se bude posouvat směrem doprava. Tento vektor je násoben hodnotou „deltaTime“ třídy „Time“, což je doba v sekundách od posledního generovaného snímku do snímku nového. V tento okamžik by se objekt pohyboval rychlostí jedné jednotky souřadnice za sekundu. Abychom tento pohyb zrychlili, násobíme jej hodnotou rychlosti získané z metody „StartFloating()“. Kdybychom hru nyní spustili, objekty mraků by existovaly navždy, což by mohlo v dlouhodobém měřítku způsobit problémy s výkonem aplikace. Je tedy nutné objekty mraků ničit pomocí metody „Destroy()“, které daný herní objekt předáváme jako parametr, když dosáhnou námi určeného bodu.

Výpis kódu 6: Metoda Update() ve scriptu CloudScript

```
void Update()
{
    if (!PauseMenu.isGamePaused)
    {
        transform.Translate(Vector2.right * Time.deltaTime * cloudSpeed);
        if (transform.position.x > cloudEndPosX)
        {
            Destroy(gameObject);
        }
    }
}
```

Zdroj: Autor (2023)

7.4 Hlavní budovy

Součástí hry jsou dvě hlavní budovy reprezentující Českou národní banku a Poslaneckou sněmovnu. Tyto budovy jsou ve scéně vytvořeny jako samostatné objekty s automaticky vytvořenou komponentou „Transform“ pro jejich umístění. Aby bylo

možné zobrazit požadovaný sprite, je nutné objektu přidat komponentu „Sprite Renderer“ a přiřadit její vlastnosti „Sprite“ požadovaný sprite. Jestliže je třeba, aby uživateli bylo dovoleno zobrazit nabídku dané budovy, pouze pokud se bude nacházet v její blízkosti, je nutno přidat objektu komponentu „Box Collider 2D“. Jestliže v nastavení komponenty nic nezměníme, bude se nyní objekt chovat jako zeď a nedovolí uživateli přes daný objekt přejít. Musíme u ní aktivovat vlastnost „Is Trigger“ která zapříčiní, že objekt se přestane chovat jako zeď, ale dovolí nám stále zjistit případnou kolizi s daným objektem. Pro rozlišení kolizí mezi objekty je třeba v inspektoru nastavit „Tag“ objektu. V našem případě se jedná o námi vytvořený „Tag“ s názvem „PS“ pro budovu Poslanecké sněmovny a „CNB“ pro budovu České národní banky. Stejně jako o objektu hlavního charakteru je dobré využít vlastnosti „Edit Collider“ pro upravení kolizního boxu daného objektu.

Pro zobrazení nabídky je třeba vytvořit script, který je přidán našemu hlavnímu charakteru jako komponenta. Ke zjištění, jestli náš hlavní charakter je v kolizi s některou z budov, lze použít metodu „OnTriggerEnter2D(Collider2D)“. Tato metoda je zavolána vždy, když kolizní box hlavního charakteru vstoupí do kolizního boxu objektu s aktivovanou vlastností „Is Trigger“. V danou chvíli můžeme rozhodnout o tom, jestli „Tag“ kolizního objektu odpovídá námi požadovanému označení budovy, jak lze vidět ve výpisu kódu 7. Na základě tohoto rozhodnutí se nám do založené proměnné „toBeEntered“ uloží hodnota ze založené proměnné „Enterable“ typu enum, která může nabývat tří hodnot: CNB, PS, None.

Výpis kódu 7: Metoda OnTriggerEnter2D()

```
void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "CNB")
    {
        toBeEntered = Enterable.CNB;
    }
    else if (collision.tag == "PS")
    {
        toBeEntered = Enterable.PS;
    }
}
```

Zdroj: Autor (2023)

Aby bylo možné rozlišit pomocí proměnné „toBeEntered“ i stav, kdy objekty již nebudou v kolizi a nemá být uživateli dovoleno nabídku spustit, lze využít metody „OnTriggerExit2D(Collider2D)“. Tato metoda je zavolána ve chvíli, kdy objekty přestanou být v kolizi. Můžeme ji tedy využít k uložení hodnoty None do proměnné „toBeEntered“.

K zapnutí nebo vypnutí nabídky můžeme využít metodu „Update()“. V ní je třeba kontroly, jestli proměnná „toBeEntered“ neodpovídá hodnotě None a zároveň, jestli byla zmáčknuta požadovaná klávesa pro otevření nabídky. Pokud je tato podmínka splněna, je třeba vnořené podmínky kontrolující proměnnou „isGamePaused“, jestli hra nebyla pozastavena, neboť nechceme uživateli dovolit otevřít nabídku budov v případě, kdy bude v herním menu. Jestliže je i tato podmínka splněna, můžeme proměnnou „isGamePaused“ nastavit na hodnotu true. Dále je třeba přistoupit ke komponentě „Rigidbody 2D“ a její vlastnosti „velocity“, kterou nastavíme na hodnotu nového nulového vektoru. Kdybychom tento krok vynechali, může nastat situace, kdy uživatel bude držet klávesu pohybu v okamžiku otevření nabídky, přičemž nebude zaznamenáno uvolnění klávesy pohybu, a hráč by nadále zůstal pro několik dalších okamžiků v pohybu. Stejně tak je důležité nastavit proměnnou „running“ v komponentě „Animator“ na hodnotu false, pomocí metody „SetBool()“, aby animace běhu nezůstala aktivní.

Nyní už zbývá pouze rozhodnout o tom, jestli se má zobrazit nabídka České národní banky či Poslanecké sněmovny. Toto rozhodnutí můžeme udělat na základně kontroly hodnoty proměnné „toBeEntered“ a aktivovat příslušný objekt pomocí metody „SetActive()“ s předaným parametrem true.

Aby měl hráč možnost z otevřené nabídky odejít stisknutím stejné klávesy, kterou použil pro vstup, můžeme vedle vnořené podmínky vytvořit podmínku „else if“, ve které je třeba kontrolovat vlastnost objektu obou nabídek „activeSelf“. Tato vlastnost je hodnoty true nebo false dle toho, jestli je objekt aktivní nebo ne. Jestliže je podmínka splněna stavem, kdy je alespoň jedna z nabídek aktivní, můžeme oba objekty nastavit na neaktivní pomocí metody „SetActive()“ s parametrem false. Splněním podmínky také víme, že proměnná „isGamePaused“ je nastavena na hodnotu true. Pro správné fungování po opuštění nabídky chceme tuto proměnou přenastavit na hodnotu false.

7.4.1 Kontextové nabídky

Nabídka je vytvořena jako nový objekt plátna, který je po spuštění hry neaktivní. Stejně jako v případě herního menu je třeba komponentě plátna „Canvas“ u vlastnosti „Render Mode“ nastavit „Screen Space – Camera“ a vlastnosti „Render Camera“ přiřadit požadovanou kameru, aby nám toto plátno vyplnilo celý prostor zobrazení kamery. Abychom pro tuto nabídku měli i nějaké pozadí, je vytvořen objekt panelu, který je podřazen objektu plátna. Tento panel můžeme pomocí automaticky přidané komponenty „Rect Transform“ ukotvit na střed plátna a zvětšit jeho velikost, aby byla s plátnem shodná. Pomocí komponenty panelu „Image“ a vlastnosti „Color“ je možné definovat barvu tohoto pozadí. Jestliže chceme využívat pouze barvu, a nikoliv sprite jako pozadí, je důležité u této komponenty odstranění přednastaveného spritu u vlastnosti „Source Image“. Pokud bychom tento sprite neodstranili, barva by u okrajů nepokrývala celou velikost panelu.

Dále je zde vytvořeno tlačítko sloužící jako alternativní způsob vypnutí nabídky. V komponentě „Button“ tohoto tlačítka u události „OnClick()“ chceme u přidaného objektu našeho herního menu zavolat metodu „Resume()“, která nabídku deaktivuje a nastaví proměnnou „isGamePaused“ na false.

Dalším zde přítomným tlačítkem je tlačítko pro zobrazení informací o nabídce a grafu. Toto tlačítko funguje na stejném principu jako většina ostatních tlačítek v hlavním a herním menu, která zobrazují požadované podnabídky.

Jelikož má uživatel možnost v každé z nabídek upravovat dva faktory hospodářské politiky dané organizace, jsou zde vytvořena pro každý faktor dvě tlačítka. Tato tlačítka mají uživateli dovolit snížit nebo zvýšit daný faktor při přechodu do dalšího časového období. Aby uživatel mohl zvolit požadovanou akci pro každý faktor, musíme místo obvyklé komponenty „Button“ přiřadit komponentu „Toggle“. Tato komponenta má výhodu díky vlastnosti „Is On“, která dokáže monitorovat, jestli bylo dané tlačítko zvoleno, a vlastnosti „Group“, která nám dovoluje stanovit skupinu tlačítek, ze kterých může být vybráno pouze jedno. Protože nechceme, aby uživatel mohl zvolit obě možnosti zároveň, můžeme tuto jednu skupinu tlačítek podřadit prázdnému objektu, kterému přidáme komponentu „Toggle Group“. Tento objekt můžeme následně u tlačítka v komponentě „Toogle“ u vlastnosti „Group“ vybrat. Díky tomu zajistíme, že pouze jedno tlačítko bude označeno za aktivní u vlastnosti „Is On“.

Problém ovšem nastává ve vizuální reprezentaci těchto tlačítek. Stejně jako u komponenty „Button“ zde máme možnost vybrat požadované barvy tlačítka při určitých stavech, jakými jsou například „Selected Color“, „Pressed Color“, „Normal Color“, ale není zde možnost volby barvy dle stavu vlastnosti „Is On“. To způsobí, že, pokud klikneme na první tlačítko, zobrazí se nám správně barva „Selected Color“, ale v okamžiku, kdy klikneme na jiné tlačítko, nově zvolené tlačítko bude považováno za vybrané a bude zobrazovat barvu „Selected Color“. Zároveň se první tlačítko vrátí do normálního stavu a bude zobrazovat bravu „Normal Color“, ačkoliv vlastnost „Is On“ je správně aktivní.

K vyřešení tohoto problému je třeba každému tlačítku přidat nově vytvořenou komponentu ve formě scriptu. Zde v metodě „Start()“ můžeme přistoupit ke komponentě „Toggle“, kterou si uložíme do proměnné „toggle“. Pro schopnost reakce na změnu hodnoty „Is On“, můžeme vytvořit metodu, která bude zavolána v případě změny této hodnoty. Toho můžeme docílit pomocí příkazu „toggle.OnValueChanged.AddListener()“, kdy jako parametr předáváme metodu „OnToggleValueChanged()“.

Jak lze vidět ve výpisu kódu 8, tato metoda je schopná dynamicky měnit barvy stavů tlačítka na základně hodnoty „isOn“. Jestliže je tato hodnota true, tlačítku bude pro jeho normální stav, i pro stav, kdy je kurzor myši nad daným tlačítkem, zvolena zelená barva. Pokud bude hodnota „isOn“ false, bude oběma stavům přiřazena barva šedá.

Výpis kódu 8: Metoda OnToggleValueChanged()

```
private void OnToggleValueChanged(bool isOn)
{
    ColorBlock cb = toggle.colors;
    if (isOn)
    {
        cb.normalColor = new Color32(15, 164, 33, 190);
        cb.highlightedColor = new Color32(15, 164, 33, 190);
    }
    else
    {
        cb.normalColor = new Color32(108, 108, 108, 255);
        cb.highlightedColor = new Color32(108, 108, 108, 255);
    }
    toggle.colors = cb;
}
```

Zdroj: Autor (2023)

7.4.2 Zobrazování grafu

Graf má za cíl vizualizovat hráčův pokrok ve hře zobrazováním hodnot míry inflace, míry nezaměstnanosti a hrubého domácího produktu v miliardách korun českých. Graf je zobrazen ve stejný okamžik jako nabídka při vstupu do budovy, a je tedy podřazen objektu panelu nabídky.

Nejprve je třeba vytvořit nový prázdný objekt, jenž je nazván „GraphContainer“, který bude mít sobě podřazeny všechny ostatní komponenty grafu. Pomocí komponenty „Rect Transform“ jej můžeme upravit k požadované velikosti. Aby šlo snadněji rozlišit prostor grafu od zbytku nabídky, je tomuto objektu vytvořeno pozadí v podobě nového prázdného objektu. Tomuto objektu lze přidat komponentu „Image“, kde vlastnosti „Source Image“ odebereme předvolený sprite a u vlastnosti „Color“ zvolíme požadovanou barvu pozadí. Pomocí komponenty „Rect Transform“ lze tento objekt pozadí ukotvit ke středu objektu „GraphContainer“ a upravit jeho velikost tak, aby odpovídala velikosti nadřazeného objektu „GraphContainer“. Jelikož je tento objekt, stejně jako další zmíněné objekty, součástí plátna, je mu také automaticky přidána komponenta „Canvas Renderer“.

Dále je třeba objektu „GraphContainer“ vytvořit nové, prázdné a neaktivní objekty, které využijeme při generování rozdělovacích čar a hodnot os po stranách grafu. Prvním takovýmto objektem je prázdný objekt nazvaný „LabelTemplateX“, který bude sloužit jako vodorovná osa s časovou reprezentací kvartálů. Tento objekt ukotvíme pomocí komponenty „Rect Transform“ k levému spodnímu rohu objektu „GraphContainer“. Abychom mohli pomocí tohoto objektu zobrazovat text, musíme mu přidat komponentu „TextMeshPro – Text“, u které odstraníme přednastavený text, a u vlastnosti „Allignment“ zvolíme „Center“ a „Top“. Dále je třeba u této komponenty změnit vlastnost „Wrapping“ na „Disabled“, aby se text v případě překročení velikosti tohoto objektu nezalamoval do nových řádků.

Obdobně vytvoříme nový neaktivní objekt, s názvem „LabelTemplateY“ pro hlavní svislou osu reprezentující hodnotu míry inflace a míry nezaměstnanosti. Zde je jedinou změnou u komponenty „TextMeshPro – Text“ a vlastnosti „Alligment“ zvolení hodnot „Right“ a „Middle“.

Tím, že na grafu má být zobrazena i hodnota hrubého domácího produktu, udávaná v miliardách korun českých, je třeba vytvořit vedlejší svislou osu, která bude schopna tyto hodnoty reprezentovat. Musíme vytvořit nový identický objekt k objektu „LabelTemplateY“, který nazveme „LabelTemplateY2“. Tento nový objekt je však nutno ukotvit pomocí komponenty „Rect Transform“ k pravému spodnímu rohu objektu „GraphContainer“.

Pro vytvoření mřížky uvnitř grafu musíme vytvořit dva objekty nazvané „DashTemplateY“ a „DashTemplateX“, jejichž rozdílem bude pouze šířka a otočení, které lze změnit u komponenty „Rect Transform“ u vlastností „Width“ a „Rotation Z“. Těmto objektům přidáme komponentu „Image“, kdy vlastnosti „Source Image“ předáme sprite který bude tvořit čerchovanou čáru. Tento sprite je z jedné poloviny bílý a z druhé průhledný. Jestliže u komponenty „Image“, a její vlastnosti „Image Type“ vybereme „Tiled“ a v inspektoru nastavení požadovaného spritu změníme vlastnost „Wrap Mode“ na hodnotu „Repeat“, dosáhneme toho, že požadovaný sprite se při změně šířky objektu neroztahuje, ale opakuje se ve své původní velikosti po celé šířce objektu. Tímto je tvořen efekt čerchované čáry.

K vyobrazení grafu musíme vytvořit novou komponentu ve formě scriptu, která pracuje se třemi samostatnými listy dat zastávajícími míru inflace, míru

nezaměstnanosti a hodnotu hrubého domácího produktu. Aby graf mohl fungovat dynamicky a mohl upravovat hodnoty os a adekvátní umístění dat na základě dat ze vstupních listů, musíme zjistit minimální a maximální hodnotu skupiny listů pro danou svislou osu. Pro zjištění těchto hodnot můžeme vytvořit novou metodu „FindMinAndMax()“, které jako parametry předáme požadované listy. Jelikož samostatnou skupinou pro nalezení minimální a maximální hodnoty je list hrubého domácího produktu, je třeba metodě nastavit výchozí hodnotu null pro druhý parametr. V metodě zakládáme dvě proměnné typu float, které mají uchovávat minimální a maximální hodnotu. Jejich výchozí hodnota je nastavena na hodnotu prvního parametru na indexu 0.

Následně můžeme cyklem foreach projít každou hodnotou prvního parametru a v případě kdy tato hodnota bude vyšší nežli naše uložené maximum nebo nižší než uložené minimum, můžeme danou hodnotu u adekvátní proměnné změnit. Jestliže druhý parametr není hodnoty null, provede se stejný cyklus i nad tímto parametrem. Aby nám při vynášení grafu graf nezasahoval přímo do okrajů námi vytvořeného prostoru pro graf, je třeba maximální hodnotu o trochu zvýšit a stejně tak minimální hodnotu snížit. Toto můžeme udělat nalezením rozdílu mezi maximem a minimem, které si uložíme do proměnné „yDifference“. V případě, že tento rozdíl bude 0, nastavíme tuto proměnnou na hodnotu 5. Následně můžeme do uloženého maxima přičíst 10 % z hodnoty proměnné „yDifference“ a stejně tak od uloženého minima odečíst. Díky tomu získáme dostatečný prostor od horní a spodní hranice grafu. Hodnoty nalezeného minima a maxima vrací funkce jako objekt „Tuple“.

Aby fungovala i vodorovná osa dynamicky a upravovala se vzdálenost mezi jednotlivými body grafu na základě jejich počtu, můžeme si do proměnné „xSize“ uložit požadovanou vzdálenost mezi jednotlivými body. Tato vzdálenost by měla vycházet z celkové šířky našeho objektu „GraphContainer“, ke které můžeme přistoupit pomocí komponenty „Rect Transform“ a její hodnoty „sizeDelta.x“, kterou si uložíme do proměnné „graphWidth“. Abychom se i zde získali dostatečný prostor od pravé hranice grafu, můžeme do proměnné „xSize“ uložit hodnotu „graphWidth“, od které odečteme hodnotu 1 a následně celek vydělíme počtem prvků v listu.

Pro vynesení os a mřížky do grafu je vytvořena metoda „ShowGraph()“, které předáváme čtyři parametry: minMaxLeft, minMaxRight, listCount a xSize. Parametry „minMaxLeft“ a „minMaxRight“ reprezentují hodnoty získané pomocí

metody „FindMinAndMax()“ pro hlavní a vedlejší svislou osu. Parametr „listCount“ je počtem prvků v listu a „xSize“ je požadovanou vzdáleností mezi body.

Jak lze vidět ve výpisu kódu 9, pro vytvoření hodnot vodorovné osy a čerchovaných svislých čar je využíváno cyklu for, který se opakuje tolikrát, kolik je prvků v listu. Proměnná „xPosition“ slouží pro určení bodu, ve kterém se má vytvořit nová instance objektu. Násobením iterační proměnné „i“ s proměnnou „xSize“ získáme lineární růst vzdálenosti mezi objekty. Hodnotu 10 přičítáme do proměnné „xPosition“ proto, aby první bod grafu nebyl přímo na jeho levé hranici.

Výpis kódu 9: Cyklus pro vodorovnou osu metody ShowGraph()

```
for (int i = 0; i < listCount; i++)
{
    float xPosition = 10 + i * xSize;

    RectTransform labelX = Instantiate(labelTemplateX);
    labelX.SetParent(graphContainer, false);
    labelX.gameObject.SetActive(true);
    labelX.anchoredPosition = new Vector2(xPosition, -10);
    labelX.GetComponent<TMP_Text>().text = "Q " + (i + 1);

    RectTransform dashX = Instantiate(dashTemplateX);
    dashX.SetParent(graphContainer, false);
    dashX.gameObject.SetActive(true);
    dashX.anchoredPosition = new Vector2(xPosition, 0);
}
```

Zdroj: Autor (2023)

Následně můžeme vytvořit instanci uloženou do proměnné „labelX“, našeho vzorového objektu „LabelTemplateX“ reprezentovaného proměnnou „labelTemplateX“ typu komponenty „Rect Transform“. Aby se tyto nově tvořené instance v hierarchii objevovaly pouze pod objektem „GraphContainer“, můžeme tento objekt nastavit jako nadřazený této instanci pomocí metody „SetParet()“. Této metodě předáváme jako první parametr nadřazený objekt. Druhý parametr je hodnoty true nebo false a udává, jestli má

být poloha podřazeného objektu relativně upravena k objektu nadřazenému. Dále je třeba nastavit instanci jako aktivní pomocí metody „SetActive()“ a nastavit novou ukotvenou pozici objektu pomocí vytvoření nového vektoru, u kterého je třeba, aby pozice X odpovídala proměnné „xPosition“ a pozice Y byla níže než je spodní hranice grafu, aby byl text dobře vidět. K vložení textu nám stačí přistoupit ke komponentě „Text Mesh Pro – Text“ a její vlastnosti „text“ přiřadit požadovanou hodnotu.

Postup vytvoření svislé čerchované čáry je téměř naprosto identický. Jelikož u objektu „DashTemplateX“ nechceme zobrazovat text, nemá tento objekt komponentu „Text Mesh Pro – Text“, ke které bychom mohli přistoupit a nastavit.

Proces vytvoření hodnot svislých os je také téměř shodný. Cyklus for zde iteruje pouze tolikrát, kolik dělících čar a hodnot chceme na grafu zobrazit. Pro umístění a zobrazení správných hodnot os zde ještě vytváříme proměnnou „normalizedValue“, která je v každém cyklu vypočítána pomocí „i * 1f / separatorCount“, kde proměnná „separatorCount“ reprezentuje hodnotu cíleného počtu iterací. Zobrazovaný text je zaokrouhleným číslem, kdy daná hodnota je vypočtena z minimální hodnoty dané osy sečtené s násobkem proměnné „normalizedValue“ a rozdílem mezi maximální a minimální hodnotou dané osy.

Pro vynesení datových bodů z listu do grafu je vytvořena metoda „CreateAndShowPoints()“. Této metodě jsou předávány čtyři parametry: valueList, minMax, xSize a color. Parametr „valueList“ je listem hodnot pro jednu křivku grafu, „minMax“ jsou hodnoty získané metodou „FindMinAndMax()“ pro požadovanou osu, parametr „xSize“ udává vzdálenost mezi body a „color“ udává barvu vykreslované křivky.

Aby bylo možné vytvořit spojovací čáru mezi dvěma body, je nutné si na začátku vytvořit nový herní objekt nazvaný „lastCircleGameObject“ s výchozí hodnotou null, do kterého budeme ukládat nejnovější vytvořený bod, jak lze vidět ve výpisu kódu 10. Jelikož chceme vytvořit nový objekt pro všechna data předaná metodě v listu, využijeme cyklus for. Stejně jako v metodě „ShowGraph()“ i zde stejným způsobem definujeme proměnnou „xPosition“ udávající pozici X daného objektu. Musíme zde ovšem definovat i proměnnou „yPosition“, aby se daný objekt správně umístil na ose Y na základě aktuálních minimálních a maximálních hodnot na dané ose. Následně můžeme vytvořit nový herní objekt „circleGameObject“ pomocí vytvořené metody „CreateCircle()“.

Tato metoda, velice podobně jako metoda „ShowGraph()“, vytvoří a vrátí nový herní objekt s komponentou „Image“, které je nastavena barva pomocí přijatého parametru „color“. Dále je přidán požadovaný sprite a ukotvena pozice pomocí komponenty „Rect Transform“ díky přijatému parametru vektoru s pozicí X a Y.

Výpis kódu 10: Vnitřní část metody CreateAndShowPoints()

```
float graphHeight = graphContainer.sizeDelta.y;
float yMin = minMax.Item1;
float yMax = minMax.Item2;

GameObject lastCircleGameObject = null;
for (int i = 0; i < valueList.Count; i++)
{
    float xPosition = 10 + i * xSize;
    float yPosition = ((valueList[i] - yMin) / (yMax - yMin)) * graphHeight;
    GameObject circleGameObject = CreateCircle(new Vector2(xPosition, yPosition), color);
    if (lastCircleGameObject != null)
    {
        CreateDotConnection(
            lastCircleGameObject.GetComponent<RectTransform>().anchoredPosition,
            circleGameObject.GetComponent<RectTransform>().anchoredPosition, color
            );
    }
    lastCircleGameObject = circleGameObject;
}
```

Zdroj: Autor (2023)

Nyní, pokud objekt „lastCircleGameObject“ není null, je vytvořeno spojení mezi posledním a novým vytvořeným bodem pomocí vytvořené metody „CreateDotConnection()“. Tato metoda přijímá jako parametry ukotvené pozice objektů „lastCircleGameObject“ a „circleGameObject“ a proměnnou „color“ využívanou ke stanovení barvy čáry mezi objekty. Obdobně jako metoda „CreateCircle()“ vytvoří nový herní objekt, kterému zde ovšem není přiřazen sprite, ale pouze barva. Velikost tohoto objektu je stanovena na základě vzdálenosti mezi metodě předanými ukotvenými pozicemi objektů. Tuto vzdálenost můžeme zjistit pomocí „Vector2.Distance()“, kdy jako parametry předáváme tyto dvě pozice.

Aby bylo možné objektu nastavit i správný úhel, je nejprve třeba vytvořit normalizovanou hodnotu mezi ukotvenými pozicemi pomocí příkazu „Vector2 dir = (dotPositionB - dotPositionA).normalized“, kde „dotPositionB“

je ukotvená pozice objektu „circleGameObject“ a „dotPositionA“ je ukotvenou pozicí objektu „lastCircleGameObject“. Tento získaný vektor s názvem „dir“ může být předán vytvořené metodě „GetAngleFromVectorFloat()“.

Jak lze vidět ve výpisu kódu 11, tato metoda vypočítá úhel mezi kladnou osou X a vstupním vektorem pomocí metody „Atan2“ z knihovny „Mathf“. Tato metoda přejímá hodnoty „dir.y“ a „dir.x“ a vrací úhel v radiánech. Tento výsledek je následně vynásoben pomocí „Mathf.Rad2Deg“ reprezentující hodnotu 180 / π, aby se úhel převedl z radiánů na stupně. Jestliže je vypočtený výsledek záporný, je k němu přičtena hodnota 360, aby výsledek byl vždy v rozsahu od 0° do 360°, a mohl díky tomu být využit pro nastavení Eulerova úhlu u komponenty „Rect Transform“.

Výpis kódu 11: Vnitřní část metody CreateAndShowPoints()

```
private static float GetAngleFromVectorFloat(Vector2 dir)
{
    dir = dir.normalized;
    float n = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
    if (n < 0) n += 360;

    return n;
}
```

Zdroj: Autor (2023)

7.5 Posun času

Hodiny slouží hráči k přesunu do dalšího časového období a jsou tedy tlačítkem, na které může uživatel kliknout. Toto tlačítko má vytvořeno samostatné plátno, obdobně jako objekty herního menu. Zde je ovšem objekt tlačítka podřazeného tomuto plátnu stále aktivní, aby na něj mohl uživatel kliknout bez potřeby provedení další akce. Aby tlačítko skutečně vypadalo jako hodiny, je třeba vytvořit jemu podřazený objekt s komponentou „Image“, které budou vlastnosti „Source Image“ požadovaný výchozí sprite předán.

Jelikož je proces přechodu do dalšího časového období pro správné fungování herních mechanik stěžejní, v případě kliknutí na tlačítko hodin je uživateli zobrazeno potvrzovací

okno, stejně jako při odchodu ze hry u herního menu. Jestliže uživatel zvolí možnost, že nechce přejít do dalšího časového období, bude toto okno zavřeno a uživatel může dále pokračovat ve hře. Pokud ovšem zvolí možnost přechodu do dalšího časového období, na hodinách se posune velká ručička o patnáct minut symbolizující přechod o jeden kvartál. Aby bylo možné této změny docílit, je vytvořena nová komponenta ve formě scriptu. Tento script obsahuje pole nazvané „spriteArray“, obsahující čtyři sprity, které budou měnit vizuální stránku hodin. Abychom mohli mezi sprity procházet cyklicky při každém potvrzení přechodu do dalšího časového období, je založena proměnná „spriteIndex“ s výchozí hodnotou 1.

Jak lze vidět ve výpisu kódu 12, v případě zavolání metody „ChangeToNextSprite()“ je nejprve zkontrolováno, jestli proměnná „spriteIndex“ není větší nebo rovna počtu prvků v poli spritů. Jestliže ano, je hodnota proměnné „spriteIndex“ nastavena na hodnotu 0. Následně je komponentě „Sprite Renderer“, zastoupené proměnnou „spriteRenderer“ nastaven požadovaný sprite z pole spritů, kdy jako index je použita proměnná „spriteIndex“. Tato proměnná je následně zvýšena o hodnotu 1, aby v případě dalšího zavolání této metody byl vyobrazen další sprite v pořadí.

Výpis kódu 12: Metoda ChangeToNextSprite()

```
public void ChangeToNextSprite()
{
    if (spriteIndex >= spriteArray.Length) spriteIndex = 0;
    spriteRenderer.sprite = spriteArray[spriteIndex];
    spriteIndex++;
}
```

Zdroj: Autor (2023)

7.6 Déšť

V případě špatného vývoje ekonomické situace ve hře se uživateli zobrazí déšť, který zároveň ztmaví obrazovku hry. Pro vytvoření tohoto efektu je potřeba vytvořit nový prázdný objekt, kterému přidáme komponentu „Particle System“ generující nové částice. Aby částice reprezentující déšť padaly směrem dolů, můžeme u této komponenty upravit vlastnost „Gravity Modifier“, jež nastavíme na hodnotu 2. K nastavení šířky a délky

generovaných částic lze u komponenty „Rect Transform“ upravovat vlastnost „Scale X“ modifikující šířku částic a vlastností „Scale Y“ nastavit jejich délku. Aby se částice negenerovaly pouze v malém bodu, můžeme u komponenty „Particle System“ zvolit kategorii „Shape“ a upravit vlastnost „Scale X“ pro roztažení plochy generující částice do požadované velikosti.

Jelikož jsme našim částicím nastavili sílu působení gravitace, částice při pádu postupně zrychlují. K dosažení realističtějšího deště je třeba posunout objekt generátoru deště nad zobrazovanou scénu. Částice tak dosáhnou relativně konstantní rychlosti při okamžiku jejich vyobrazení. Pro zvýšení počtu generovaných částic můžeme u komponenty „Particle System“ v kategorii „Emission“ u vlastnosti „Rate over Time“ tuto hodnotu navýšit na požadované množství.

Pro úpravu barvy částic můžeme vytvořit nový materiál, kterému u vlastnosti „Shader“ zvolíme možnost „Sprite“. Následně můžeme změnit vlastnost „Tint“ na námi požadovanou barvu. Tento materiál můžeme předat komponentě „Particle System“ v kategorii „Renderer“ vlastnosti „Material“.

Nyní je třeba definovat dobu života padající částice, aby zbytečně dlouho nepadaly pod naši scénu nebo naopak nebyly ničeny v polovině zobrazované scény. K tomu můžeme adekvátně upravit hodnotu vlastnosti „Start Lifetime“ komponenty „Particle System“. V dané komponentě můžeme ještě nastavit minimální a maximální velikost generovaných částic změnou vlastnosti „Start Size“ na „Random Between Two Constants“, kde můžeme následně definovat minimální a maximální měřítko částic.

Pro dosažení efektu ztmavení obrazovky lze vytvořit plátno s pozadím obdobně jako u herního menu. Barvě pozadí je zde nutné nastavit určitou průhlednost k zachování viditelnosti zbytku hry.

7.7 Úvodní instrukce

Úvodní instrukce ve hře slouží k informování uživatele o základních principech fungování herních mechanik, jež jsou pro uživatele relevantní. Jedná se například o instrukce pro pohyb hlavní postavy, způsob otevření herního menu, otevření nabídky budov nebo o instrukce pro přesun do dalšího časového období.

Instrukce jsou uživateli předávány postupně na základě provedení předešlých pokynů. Aby bylo možné instrukční okno zobrazit, je vytvořen nový objekt plátna se stejnými parametry jako v předešlých případech. Následně je plátnu vytvořen podřízený nový prázdný objekt, který využijeme pro ovládání jednotlivých instrukcí. Jednotlivé instrukce jsou založeny jako objekty se spritem pozadí pro dané instrukce a specifickým textem pro každý pokyn. K jejich ovládání je vytvořena nová komponenta v podobě scriptu.

V metodě „Update()"scriptu je třeba nejprve vytvořit podmínku kontrolující, zdali již nebyly zobrazeny všechny požadované instrukce. Toho můžeme docílit založením proměnné nazvané „popUpIndex" reprezentující hodnotu právě aktivní instrukce. Jestliže je hodnota menší nebo rovna délce pole, v němž jsou všechny instrukce uloženy, můžeme pomocí cyklu for projít všemi prvky pole instrukcí. Je-li hodnota proměnné „popUpIndex" a iterační proměnná „i" cyklu for, budou shodné a zároveň nebude aktivní objekt herního menu nebo některá z nabídek budov, můžeme danou instrukci z pole instrukcí na indexu iterační proměnné „i" aktivovat pomocí metody „SetActive()". Pokud nebude podmínka splněna, danou instrukci deaktivujeme.

Pro zvyšovaní proměnné „popUpIndex" je uvnitř první podmínky vytvořen switch. Zde je pro každou potenciální hodnotu proměnné „popUpIndex" vytvořen případ, jenž kontroluje podmínku odpovídající dané instrukci. Jestliže bude podmínka splněna, je do proměnné „popUpIndex" přičtena hodnota 1, a tím zobrazena následující instrukce.

Aby instrukce byly zobrazeny vždy přímo nad hlavním charakterem, je v metodě „Update()" vždy nastavena jejich pozice X na pozici X hlavního charakteru a pozice Y je adekvátně zvýšena, aby instrukce hlavní charakter nezakrývaly.

7.8 Tvorba grafických prvků

Pro tvorbu grafických prvků, ve stylu pixel art, byl zvolen software Krita[4], neboť je dostupný zcela zdarma a poskytuje všechny potřebné nástroje k tvorbě požadované grafiky. V nastavení softwaru Krita v kategorii „Zobrazit" a následně „Nastavení mřížky" můžeme nastavit barvu a zobrazení mřížky pixelů. Díky tomu jsou individuální pixely při tvorbě grafiky samostatně označeny, což nám dovoluje snáze s nimi pracovat.

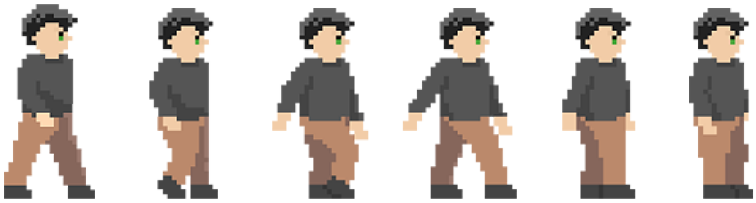
[4] Autor Matthias Ettrich – www.krita.org

Většina grafických prvků hry byla vytvořena v samostatných vrstvách jednoho dokumentu. Díky tomu vidíme jednotlivé prvky jako celek a můžeme posoudit vhodnost jednoho prvku v porovnání s ostatními. Dosáhneme tak větší koherence grafických prvků napříč projektem.

7.8.1 Postava hlavního charakteru

Aby hlavní charakter vypadal ve hře přirozeněji, bylo pro jeho animaci chůze vytvořeno šest snímků, které při přehrání tvoří dojem chůze. Posledním snímkem je stojící postava, kterou lze využít jako zobrazovaný sprite postavy, pokud není v pohybu.

Obrázek 2: Snímky chůze postavy hlavního charakteru



Zdroj: Autor (2023)

7.8.2 Hlavní budovy institucí

Jako předlohy budov institucí ve hře České národní banky a Poslanecké sněmovny sloužily skutečné fotky budov. Vytvořené budovy, které lze vidět na obrázku 2, byly upraveny svými rozměry, aby lépe odpovídaly požadavkům hry. Zároveň byl budovám přidán šedivý odstín do oken nižších pater, který ve výsledné hře simuluje odraz silnice.

Obrázek 3: Budovy České národní banky a Poslanecké sněmovny

Zdroj: Autor (2023)

7.8.3 Pozadí hry

Pozadí, včetně ostatních grafických prvků, má za cíl vytvořit vhodné propojení mezi všemi zobrazovanými objekty. Proto byla vytvořena také řada objektů, které by měly herní prostředí vhodně doplnit. Jedná se například o stromy, keře, mraky, lavičky a další budovy v pozadí, jenž v kombinaci se silnicí, trávou a oblohou tvoří plný a různorodý svět.

7.9 Tvorba zvukových prvků

Aby hra nepůsobila příliš stroze, byla pro ni vytvořena podkladová hudba a zvukové efekty deště a otevírání vrzajících dveří.

7.9.1 Podkladová hudba

Pro tvorbu podkladové hudby byl vybrán volně dostupný software LMMS[5], obsahující řadu předpřipravených vzorků, zvuků a syntezátorů, které jsou také volně dostupné k užití.

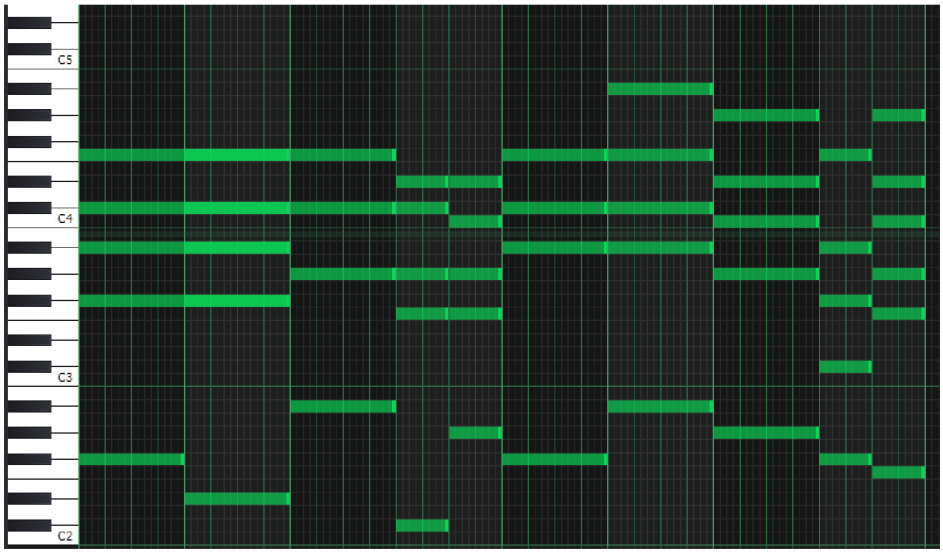
Výsledná podkladová hudba by měla na uživatele působit klidným dojmem a je složena tak, aby šla přehrávat v nekonečné smyčce. Jelikož hudba hraje v pozadí průběhu celé hry, je důležité zvolit příjemnou podkladovou zvukovou stopu vhodnou pro dlouhý poslech.

[5] Autoři: Paul Giblock, Tobias Junghans – www.lmms.io

Pomocí nástroje „Song-Editor“ obsaženého v programu LMMS můžeme přidávat jednotlivé vzorky nástrojů u kterých lze pomocí virtuálních kláves vytvořit melodickou linku. Díky syntezátoru ZynAddSubFX jsme následně schopni vzorky upravovat a tvořit nové.

Jak lze vidět na obrázku 4, finální vytvořená hudba se skládá z deseti na sebe navazujících akordů, jejichž zvuk vychází ze vzorku „Soft Piano“ obsaženého v programu LMMS. Vzorek byl následně upraven pomocí syntezátoru ZynAddSubFX, kde mu byla upravena výška, ostrost, doba nástupu a filtr frekvence pro aplikování efektu rezonance.

Obrázek 4: Virtuální klávesy s vytvořenými akordy



Zdroj: Autor (2023)

K této lince byla přidána i linka reprezentující zvuky bubnů vytvořená pomocí nástroje „Beat+Bassline Editor“. V nástroji můžeme zvolit moment, kdy má buben zaznít, bez nutnosti upravovat jeho frekvenci. Pro bubny byly zvoleny vzorky „Hat“ a „Kick“ upravené tak, aby nezněly příliš ostře a více se hodily k melodické lince. Nad celou skladbou je vytvořen také efekt dozvuku (Reverb), přidávající hudbě pocit prostorovosti.

7.9.2 Zvukové efekty

Pro nahrávání zvukových efektů byl využit volně dostupný software Audacity. Jako nahrávací zařízení byl použit kondenzátorový mikrofon STC-1[6] a jako zvukové rozhraní pro propojení s počítačem byl použit mixpult XENYX QX2442 USB[7]. Díky tomu, že tento mixpult disponuje USB rozhraním pro propojení s počítačem, je převod analogového zvuku na zvuk digitální obstaráván přímo mixpultem, což napomáhá k odstranění zvukových defektů. Zároveň pomocí mixpultu můžeme v reálném čase upravovat nahrávaný zvuk a použitím frekvenčního filtru odstranit hluboké tóny, které jsou při nahrávání venkovních zvuků nejčastěji způsobované foukáním větru do mikrofonu.

Při nahrávání je důležité snížit na naprosté minimum zvuky rušící finální nahrávku. Proto nahrávání zvuků hry probíhalo výhradně v noci, kdy se okolní zvuky, jakým je například projíždějící auto, vyskytovaly pouze zřídka.

[6] Sontronics/Omnisonic Internationa Ltd., Poole, Spojené Království
[7] BEHRINGER Spezielle Studiotechnik GmbH, Willich, Německo

8 Závěr

V této práci se podařilo vytvořit komplexní uživatelské prostředí hry simulující hospodářský cyklus. V prostředí herního enginu Unity byly například vytvořeny mechaniky pro funkční navigaci v úvodním a herním menu, generaci a pohyb mraků, generaci částic deště, pohyb a animace herní postavy, zobrazování úvodních instrukcí, nabídky pro volbu požadovaných ekonomických faktorů či graf generovaný z listu dat.

Jako grafický styl hry byl zvolen pixel art, v němž byly následně vytvořeny všechny grafické assety hry. Pro tvorbu grafických assetů byl zvolen program Krita. Aby hra na uživatele nepůsobila příliš stroze, krom hlavních budov institucí a postavy hráče byla vytvořena řada dalších prvků doplňujících herní prostředí. Jedná se například o stromy, keře, rostliny, mraky či budovy pozadí. Zároveň byly také vytvořeny speciální prvky zobrazované pouze v okamžiku, kdy je ve hře špatná ekonomická situace.

K doplnění celkové atmosféry hry byla vytvořena podkladová hudba za využití programu LMMS. Dále byly pomocí programu Audacity nahrány a upraveny zvukové efekty pro otevírání dveří a padající déšť.

Na základě těchto skutečností lze považovat cíl práce za naplněný.

I. Summary and keywords

This thesis focusses in detail on the processes necessary to create a user interface of a 2D game created in the Unity game engine using the C# language. The work also includes the creation of game graphics, textures, sounds, and the process of data visualisation in graph. In the first part of this work, the individual tools that can be used for the purposes of this work are explained and described. In the practical part of this work, the selected tools are used to create a game that brings the principles and processes of economic policy closer to the user in a friendly and easy-to-understand environment, where the user can move between individual institutions and make changes in economic policy.

Key words: application, Unity game engine, C# programming language, pixel art, 2D, educational game, sound creation, economic policy

II. Seznam literatury

About Minecraft. (2023). Získáno z Minecraft website: https://www.minecraft.net/en-us/about-minecraft

Adams, E. (2010). *Fundamentals of Game Design: Fundamentals of Game Design_2*. New Riders.

Addo, A. (2017, srpen 31). Pretty Pixels—The importance of visuals in game design. Získáno 9. duben 2023, z Cube website: https://medium.com/the-cube/pretty-pixel-the-importance-of-visuals-in-game-design-5f3ae148a41e

Barnes, S. (2022, březen 23). How the Humble Pixel Became a Building Block To Groundbreaking Art. Získáno 21. březen 2023, z My Modern Met website: https://mymodernmet.com/what-is-pixel-art/

Bartlett, B., & Bartlett, J. (1999). *On Location Recording Techniques*. New York: Routledge. https://doi.org/10.4324/9780080513041

Bellotti, F., Berta, R., De Gloria, A., Lavagnino, E., Dagnino, F., Ott, M., … Mayer, I. S. (2012). Designing a Course for Stimulating Entrepreneurship in Higher Education through Serious Games. *Procedia Computer Science*, *15*, 174–186. https://doi.org/10.1016/j.procs.2012.10.069

Clark, Smith, Killingsworth, S. S., & Douglas. (2016). Digital Games, Design, and Learning: A Systematic Review and Meta-Analysis. *Review of Educational Research*, *86*(1), 79–122. https://doi.org/10.3102/0034654315582065

Elias, D. (2021, květen 18). Unity Visual Scripting, Part 5 – Control Flow | Unity Tutorial. Získáno 25. březen 2023, z NotSlot website: https://notslot.com/tutorials/2021/05/visual-scripting-101-getting-started

Gad, S. (2022, leden 29). 5 Lessons in Finance and Investing From Monopoly. Získáno 9. duben 2023, z Investopedia website: https://www.investopedia.com/articles/basics/12/lessons-monopoly-teaches.asp

Gee, J. P. (2004). *WHAT VIDEO GAMES HAVE TO TEACH US ABOUT LEARNING AND LITERACY* (1. vyd.). New Yokr: Palgrave Macmillan.

Godot Engine. (2023, březen 1). Godot 4.0 sets sail: All aboard for new horizons. Získáno 26. březen 2023, z Godot Engine website: https://godotengine.org/article/godot-4-0-sets-sail/

Gregersen. (2022, říjen 7). Raster graphics | Definition, Examples, Advantages, & Facts | Britannica. Získáno 23. březen 2023, z Britannica website: https://www.britannica.com/technology/raster-graphics

Gruyer, N., & Toublanc, N. (b.r.). Who we are? Získáno 24. březen 2023, z https://economics-games.com/gameswho

Halpern, J. (2019). *Developing 2D Games with Unity: Independent Game Programming with C#* (1. vyd.). New York. Získáno z https://www.oreilly.com/library/view/developing-2d-games/9781484237724/

Holmes, L. (2022, březen 10). Why Music Is So Important In Gaming. Získáno 9. duben 2023, z TotalNtertainment website: https://www.totalntertainment.com/features/why-music-is-so-important-in-gaming/

Konfršt, J. (2019, leden 23). Adventura The Oregon Trail se vrací jako retro handheld. Získáno z Vortex website: https://www.vortex.cz/adventura-the-oregon-trail-se-vraci-jako-retro-handheld/

Martin, J. (2020, říjen 20). What is a Game Engine? | University of Silicon Valley. Získáno 25. březen 2023, z https://usv.edu/blog/what-is-a-game-engine/

Nugent, J. (2020, prosinec 4). What is a synthesizer, how it works, types and uses. Získáno 10. duben 2023, z Higher Hz website: https://higherhz.com/what-is-synthesizer/

Nugent, J. (2021, leden 21). What is a MIDI controller/keyboard, what it does, and why you need one? Získáno 10. duben 2023, z Higher Hz website: https://higherhz.com/what-is-midi-controller-keyboard/

Rogmans, T. (2018, srpen 15). Macroeconomics Simulation: Econland. Získáno 24. březen 2023, z Harvard Business Publishing website: https://hbsp.harvard.edu/product/8725-HTM-ENG?Ntt=

Rogmans, T. (2022, říjen 1). The impact of an online macroeconomics simulation game on student engagement and performance [MPRA Paper]. https://doi.org/10/MPRA_paper_115347.pdf

Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on Game Design*. New Riders.

Saunders, K., & Novak, J. (2012). *Game Development Essentials: Game Interface Design*. Cengage Learning.

Sinicki, A. (2021, březen 20). What is Unity? Everything you need to know. Získáno 25. březen 2023, z Android Authority website: https://www.androidauthority.com/what-is-unity-1131558/

Todd, D. (2023). Vector art: What is vector art? | Adobe. Získáno 23. březen 2023, z https://www.adobe.com/creativecloud/illustration/discover/vector-art.html

Tucker, T., & Pearce, A. (2015, prosinec 9). How Do People Play The Trading Game? Získáno 24. březen 2023, z Bloomberg.com website: http://www.bloomberg.com/features/2015-stock-chart-trading-game/analysis/

Unity Technologies. (2018, červenec 31). Unity - Manual: Prefabs. Získáno 20. březen 2023, z Docs.unity3d website: https://docs.unity3d.com/Manual/Prefabs.html

Unity Technologies. (2022, říjen 13). Unity Editor Software Terms. Získáno 26. březen 2023, z https://unity.com/legal/editor-terms-of-service/software

Unity Technologies. (2023a). Compare Unity plans: Pro vs Plus vs Free. Choose the best 2D - 3D engine for your project! - Unity Store. Získáno 26. březen 2023, z https://store.unity.com/compare-plans

Unity Technologies. (2023b, duben 7). Unity - Manual: Animator Controller. Získáno 25. březen 2023, z https://docs.unity3d.com/Manual/class-AnimatorController.html

What is Kahoot!? (2023). Získáno z Kahoot website: https://kahoot.com/what-is-kahoot/

Wirtz, B. (2019, listopad 24). Unity Engine vs. Godot: The Ultimate Game Engine Showdown. Získáno 26. březen 2023, z https://www.gamedesigning.org/engines/unity-vs-godot/

YeeOfficer. (2020, prosinec 2). By adding a splash screen on the free version, Unity creates a massive stigma around their engine. [Reddit Post]. Získáno 26. březen 2023, z R/Unity3D website: www.reddit.com/r/Unity3D/comments/k54smg/by_adding_a_splash_screen_on_the_free_version/

III. Seznam obrázků

IV. Seznam tabulek

V. Seznam výpisů kódu

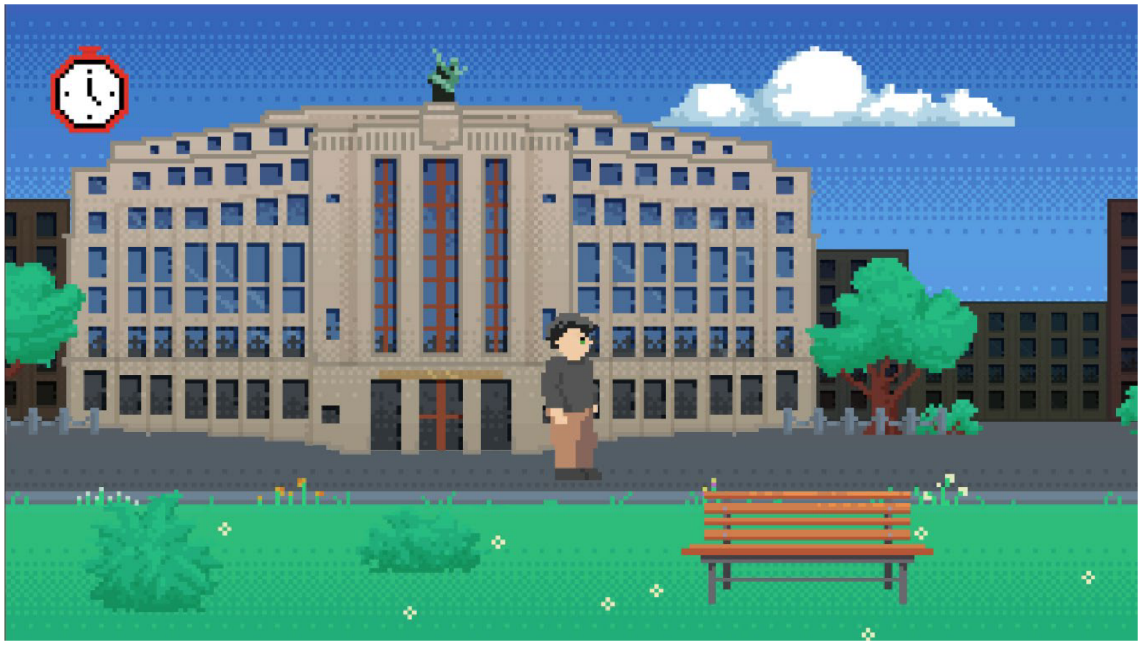
VI. Seznam příloh

VII. Přílohy

Příloha 1: Úvodní hlavní menu hry

Zdroj: Autor (2023)

Příloha 2: Prostředí hry



Zdroj: Autor (2023)

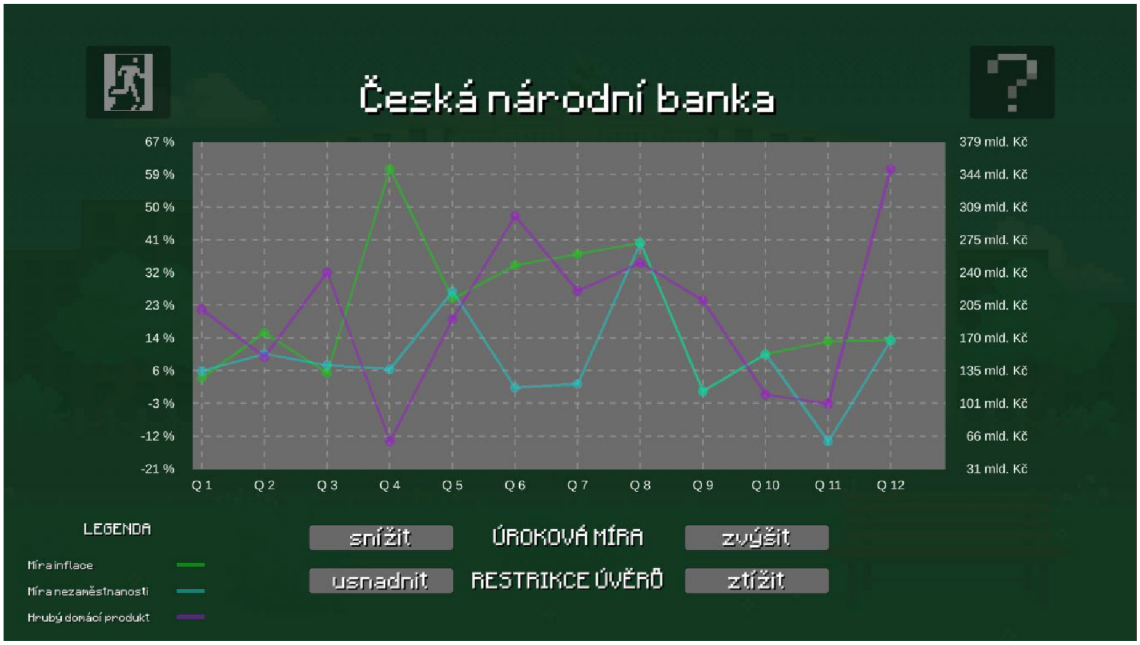
Příloha 3: Úvodní instrukce při spuštění hry

Zdroj: Autor (2023)

Příloha 4: Zobrazení hry při špatné ekonomické situaci

Zdroj: Autor (2023)

Příloha 5: Nabídka České národní banky s grafem náhodných dat



Zdroj: Autor (2023)

Příloha 6: Předloha pro zobrazení textu ekonomické události

Zdroj: Autor (2023)