Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Diskrétní simulace za použití knihovny SimPy

2026

Panovský Tomáš

Vedoucí práce:
Mgr. Jan Laštovička, Ph.D.

Studijní program: Informační technologie,
prezenční forma

Bibliografické údaje

Autor:	Panovský Tomáš
Název práce:	Diskrétní simulace za použití knihovny SimPy
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2026
Studijní program:	Informační technologie, prezenční forma
Vedoucí práce:	Mgr. Jan Laštovička, Ph.D.
Počet stran:	58
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Panovský Tomáš
Title:	Thesis title
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2026
Study program:	Information Technologies, full-time form
Supervisor:	Mgr. Jan Laštovička, Ph.D.
Page count:	58
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Tato práce se zabývá principy diskrétní simulace a využitím těchto principů při modelování hudebního festivalu za účelem analýzy vytížení jednotlivých obslužných míst festivalového areálu. Součástí práce je simulační program implementovaný v Pythonu, který využívá knihovnu SimPy pro diskrétní simulaci a Tkinter pro vizualizaci a tvorbu grafického editoru festivalového areálu. Program umožňuje nastavit parametry simulace, vytvářet rozložení areálu a sestavit harmonogram vystupujících kapel. Výstupem simulace jsou podrobné výpisy událostí návštěvníků, statistiky o vytížení obslužných míst a výpisy z celkového průběhu festivalu.

Synopsis

This thesis examines the principles of discrete-event simulation and applies them to the modelling of a music festival with the aim of analysing the workload of individual service points within the festival grounds. The work includes a simulation program implemented in Python, using the SimPy library for discrete-event simulation and Tkinter for visualisation and for creating a graphical editor of the festival layout. The program allows users to configure simulation parameters, design the festival area, and create a schedule of performing bands. The simulation outputs detailed visitor logs, statistics on the utilisation of service points, and records of the overall course of the festival.

Klíčová slova: simulace, SimPy, vizualizace, organizace, fronty, festival

Keywords: simulation, SimPy, visualization, organization, queues, festival

Děkuji vedoucímu práce Mgr. Janu Laštovičkovi, Ph.D. za ochotu, velmi cenné rady, trpělivost a vedení mé bakalářské práce.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Organizace rozsáhlých hudebních festivalů a kulturních akcí představuje pro pořadatele komplexní úkol, který zahrnuje správné rozmístění obslužných míst a rozhodování o jejich kapacitě, tedy kolik vstupů, stánků či dalších služeb bude v areálu umístěno a kolik návštěvníků dokáže každý z nich v daný okamžik obsloužit. Nedostatečné plánování může vést k dlouhým frontám a následnému zhoršení celkového zážitku účastníků.

Tyto skutečnosti mě přivedly k úvaze, zda by bylo možné podobné situace předem modelovat a tím organizátorům usnadnit rozhodování o vhodném počtu zdrojů. Snaha podpořit efektivnější organizační rozhodování představuje hlavní motivaci této práce, jejímž cílem má být jednoduchý simulační nástroj umožňující předvídat vznik front a vyhodnocovat dopady různých organizačních rozhodnutí.

Simulace představuje užitečný prostředek pro zkoumání situací, které jsou obtížné nebo nákladné na testování v reálném provozu. Umožňuje bezpečně experimentovat s různými variantami uspořádání areálu či měnit počty obslužných míst a následně pozorovat dopady těchto změn bez rizika negativních následků. Díky tomu lze lépe odhalit kritická místa v organizaci.

Práce obsahuje simulační program, který umožňuje nastavit parametry simulace, vytvořit mapu festivalového areálu, sestavit program vystupujících kapel a následně nasimulovat průběh celého festivalu. Výstupem programu jsou podrobné výpisy událostí jednotlivých návštěvníků i celkového průběhu akce a statistiky o vytížení jednotlivých obslužných míst. Průběh simulace je možné sledovat v reálném čase přímo v programu, nebo ji spustit ve zrychleném režimu a ihned přejít k výsledkům.

Teoretická část práce pokrývá základy diskrétní simulace vysvětlené ve druhé kapitole. Třetí kapitola popisuje použité technologie a zejména knihovnu SimPy a její klíčové prvky využité při implementaci. SimPy je Python knihovna pro diskrétní simulace založené na událostech. Čtvrtá kapitola práce se věnuje technické dokumentiaci a je zaměřena především na návrh modelu festivalu, definici datových struktur, logiku návštěvníků a detailní popis architektury aplikace. Součástí práce je také uživatelský pohled na aplikaci v páté kapitole, kde je podrobně popsáno její ovládání, nastavení simulace i práce s grafickým editorem festivalového areálu. Závěrečná část se věnuje výsledkům simulace a možnostmi dalšího rozšíření aplikace.

2 Diskrétní simulace

V této kapitole vycházející z publikace [1] si představíme principy diskrétní simulace. Zaměříme se zejména na pojem systému, jeho stavových proměných a simulačního modelu, který tvoří základ pro další části této práce.

2.1 Systém

Reálnou skutečnost, kterou chceme analyzovat, označujeme jako *systém*. Analyzovat znamená sledovat, jak se různé části systému chovají v čase a vyvozovat závěry o efektivitě, kapacitě nebo chování systému.

Na systémy můžeme nahlížet jako diskrétní nebo spojité, přičemž záleží na úhlu pohledu a vlastnostech, které v systému převažují. *Diskrétní systém* je takový, u kterého se stavové proměnné mění pouze v diskrétních časových okamžicích. Oproti tomu *spojitý systém* je takový systém, u kterého se stavové proměnné mění plynule v čase.

Stav systému je definován jako soubor stavových proměnných, které jsou nezbytné k popisu systému v libovolném okamžiku. Stavové proměnné představují minimální množinu veličin potřebných k zachycení aktuálního stavu systému vzhledem k cíli studie.

Každý systém lze chápat jako soubor vzájemně propojených prvků, které společně ovlivňují jeho chování. Aby bylo možné systém lépe pochopit, je vhodné rozdělit jej na základní komponenty, které popisují jeho strukturu a dynamiku. Mezi tyto komponenty patří entity, jejich atributy, aktivity a události. *Entita* je objekt zájmu v systému. V našem případě mohou být entitami například návštěvníci festivalu, stánky s občerstvením nebo kapely. *Atribut* je vlastnost entity, například statistiky návštěvníka informující o jeho aktuálním stavu hladu či únavy. *Aktivita* představuje časově omezenou činnost entity, například čekání ve frontě, sledování koncertu nebo nákup jídla. *Událost* je okamžitá změna stavu systému.

2.2 Úvod do simulace

Simulace je napodobení systému v čase. Cílem je vytvořit umělou historii stavu daného systému a následně pozorovat tuto uměle vytvořenou historii za účelem vyvození závěrů o provozních vlastnostech systému.

Chování systému vyvíjejícího se v čase se zkoumá pomocí simulačního modelu. Model lze následně využít k analýze hypotetických variant provozu a k posouzení dopadů změn jednotlivých parametrů na chování reálného systému. Například můžeme zkoumat, zda je daný počet stánků s občerstvením dostatečný pro obsloužení všech návštěvníků festivalu bez dlouhých front.

Možné změny systému pak lze nejprve nasimulovat, aby bylo možné předpovědět jejich dopad na výkonnost systému. Simulace může být také použita ke studiu systémů ve fázi návrhu, ještě před jejich samotnou realizací.

2.3 Model systému

Model je definován jako reprezentace systému za účelem jeho studia. Pro většinu studií je nutné zvažovat pouze ty aspekty systému, které ovlivňují problém, který je předmětem zkoumání.

Modely lze klasifikovat jako statické nebo dynamické, deterministické nebo stochastické a diskrétní nebo spojité. Statický simulační model reprezentuje systém v určitém časovém okamžiku, zatímco dynamické modely reprezentují systémy, jak se mění v čase. Deterministické simulační modely neobsahují žádné náhodné prvky, a to ani ve vstupních datech, ani v průběhu samotné simulace. Při opakovaném spuštění se stejnými vstupy poskytují vždy totožný průběh i výsledek simulace. Stochastické modely naproti tomu obsahují jeden nebo více náhodných prvků, které mohou vstupovat do simulace jak na jejím začátku, tak v jejím průběhu, například při generování časů událostí nebo rozhodování o chování entit. Díky tomu lépe vystihují systémy, jejichž chování je ovlivněno náhodou. Výsledky takových simulací nejsou jednoznačné, ale mají pravděpodobnostní charakter. Diskrétní a spojité modely jsou definovány obdobně jako u systémů.

Pro studium hudebního festivalu použijeme diskrétní, dynamický a stochastický model. Diskrétní model, protože stavové proměnné se mění pouze v konkrétních okamžicích. Dynamický model, protože sleduje vývoj systému v čase během celé doby trvání festivalu. Stochastický model, protože některé vstupy, například doba čekání u stánku, jsou náhodné.

3 Použité technologie

V této kapitole si představíme technologie a knihovny využité při vývoji aplikace. Pro implementaci simulačního modelu byl zvolen programovací jazyk Python. Hlavními důvody této volby jsou jeho vysoká úroveň abstrakce, široká dostupnost knihoven a rozsáhlá uživatelská komunita, která poskytuje podrobnou dokumentaci. Samotná simulace je realizována pomocí knihovny SimPy, která je určena pro diskrétní simulaci a poskytuje nástroje pro práci s procesy, událostmi a zdroji. V této práci tvoří SimPy největší část implementace, a proto bude podrobněji představena v následujících podkapitolách.

Grafické uživatelské rozhraní aplikace je vytvořeno pomocí knihovny Tkinter, která je součástí standardní instalace Pythonu a poskytuje základní widgety a nástroje pro tvorbu grafického uživatelského rozhraní. Pro modernější vzhled a jednodušší práci se styly byla použita nadstavba CustomTkinter, která rozšiřuje možnosti Tkinteru o vizuálně atraktivnější prvky jako jsou například tlačítka se zaoblenými rohy. Ikony a grafické prvky rozhraní jsou zpracovány pomocí knihovny Pillow, která umožňuje manipulaci s obrázky, jejich načítání a zobrazování v GUI. Většina ikon použitých v aplikaci byla převzata z webu [2]. Doplňkové ikony a také obrázek použitý na pozadí aplikace byly vytvořeny pomocí generátoru obrázků `Microsoft Copilot`.

3.1 Knihovna Simpy

Informace v této podkapitole vychází z oficiální SimPy dokumentace [3]. Knihovna SimPy je založena na využití generátorů a příkazu `yield`. Generátory předávají řízení simulace simulačnímu prostředí prostřednictvím událostí, na jejichž splnění proces čeká, jak si ukážeme v následujících kapitolách.

3.1.1 Generátory v Pythonu

Iterátor [4] je objekt, který umožňuje postupné získávání hodnot bez nutnosti mít všechny hodnoty uložené v paměti. Iterátor si pamatuje svůj aktuální stav a při každém volání funkce next() vrací další prvek.

Generátor je speciální forma iterátoru, který obsahuje v těle funkce příkaz `yield`. Příkaz `yield` umožňuje generátoru postupně produkovat jednotlivé prvky posloupnosti a může teoreticky produkovat i nekonečnou posloupnost dat. Posloupnost zde znamená řadu hodnot, které generátor postupně poskytuje. Příkaz produkující prvek posloupnosti nabývá tvaru: `yield element`. Po provedení příkazu yield se generátor pozastaví a vrátí hodnotu specifikovanou příkazem `yield`. Při dalším volání pokračuje ve vykonávání od místa, kde byl přerušen.
Napřed pouze vytvoříme generátor i:

```
i = get_numbers()
```

```
1 def get_numbers():
2   i = 0
3   while True:
4     yield i
5     i = i + 1
```

Zdrojový kód 1: Příklad generátoru

Další prvek můžeme vyžádat funkcí next. Dalším prvkem posloupnosti bude hodnota určená příkazem yield. Tedy:

```
next(i)
```

V tuto chvíli bude v proměnné i uložena 0. Tělo generátoru se začne vykonávat od pozastaveného místa až po příkaz yield. Vykonávání těla generátoru je pozastaveno na řádku:

```
i = i + 1
```

Popsaným způsobem získáme další hodnoty z generátoru, tedy při dalším volání `next(i)` bude v proměnné `i` 1, poté 2, a tak dále.

V kontextu diskrétní simulace v knihovně SimPy jsou generátory využity k modelování aktivit, které představují chování jednotlivých entit. Každý příkaz `yield` v generátoru odpovídá předání řízení simulátoru a obvykle vrací objekt typu `Event`, který reprezentuje událost, na jejíž dokončení aktivita čeká.

3.1.2 Základní principy SimPy

Základní komponenty SimPy jsou *prostředí simulace* (jenž je v SimPy označováno jako `Environment`), *události* a *procesní funkce*. Smyslem prostředí je řídit celou simulaci, udržovat simulační čas a organizovat události. Události udávají, kdy má dojít ke změně stavu a společně tvoří časový plán simulace. Procesní funkce jsou python generátorové funkce, které generují události. Smyslem procesních funkcí je popisovat chování entit v simulaci. Na každou procesní funkci můžeme nahlížet jako na aktivity dané entity. Voláním procesní funkce vzniká entita, jejíž chování popisuje iterátor funkcí vrácený. Jeden z argumentů volání procesní funkce je vždy prostředí. Instance procesní funkce běžící v simulačním prostředí v kontextu SimPy a programového řízení simulace nazýváme proces.

Prostředí simulace řídí průběh simulační času a spravuje seznam plánovaných událostí. Simulační čas je čas, ve kterém simulace probíhá, nikoliv reálný čas. V SimPy je tento čas oproti reálnému času bez konkrétní jednotky a probíhá diskrétně. To znamená, že se čas posouvá od jedné události k další události. Hodnoty času však nejsou omezeny na celá čísla. SimPy používá běžné číselné typy jazyka Python, tedy i hodnoty typu float. Simulační čas tak může nabývat hodnot jako například 0, 1, 2, ale také 1.5 nebo 3.25. Jednotky času si můžeme určit sami dle kontextu simulace. Mohou to být například hodiny, minuty či

sekundy. V této práci je jedna jednotka času rovna jedné minutě. Standardně čas začíná hodnotou 0. Aktuální čas simulace můžeme získat zasláním zprávy `env.now()`. Každá událost má přiřazený čas, ve kterém nastane. Prostředí tedy vždy zpracuje událost naplánovanou na aktuální čas.

Jednotlivé aktivity reprezentované procesní funkcí mohou probíhat nezávisle na sobě. To, že aktivita probíhá nějakou dobu, lze v SimPy simulovat události `environment.timeout(time)`, která procesní funkci na stanovený čas pozastaví. Během doby, kdy je procesní funkce pozastavena, může prostředí vykonávat jiné události naplánované na aktuální čas simulace. K pozastavení procesní funkce dochází ve chvíli, kdy entita provádí časově omezenou aktivitu. Pozastavený proces tedy představuje období, během kterého simulovaná entita vykonává danou aktivitu. Procesní funkce lze pozastavit i dalšími typy událostí, které budou vysvětleny později.

Abychom ilustrovali, jak SimPy pracuje s procesními funkcemi a jak mezi nimi přepíná pomocí událostí, si představme jednoduchý scénář se dvěma roboty, kteří se pohybují v různých směrech. Každý robot představuje samostatnou entitu se svou vlastní aktivitou, která nějakou dobu trvá. Právě tuto dobu můžeme v SimPy modelovat událostí `env.timeout()`. V následujícím příkladu vznikne zavoláním procesní funkce `robot` entita `Robot A` a entita `Robot B`. Tělo procesní funkce zařídí, že vzniklý robot po náhodně dlouhou dobu půjde směrem vlevo nebo vpravo a poté se na jednu časovou jednotku zastaví. Pro určení doby, jak dlouho se roboti budou pohybovat použijeme modul `random` ze standardní knihovny Pythonu. Prostředí simulace mezi těmito dvěma aktivitami automaticky přepíná vždy ve chvíli, kdy některý z robotů čeká na dokončení události `timeout`.

```
1  env = simpy.Environment()
2
3  def robot(env, name, direction):
4     while True:
5        print(f"{env.now}: {name} se začal pohybovat směrem
             {direction}")
6        yield env.timeout(random.randint(1, 3))
7        print(f"{env.now}: {name} se zastavil a rozhlíží se")
8        yield env.timeout(1)
9
10 env.process(robot(env, "Robot A", "vpravo"))
11 env.process(robot(env, "Robot B", "vlevo"))
12 env.run()
```

Zdrojový kód 2: Příklad procesní funkce

Nejprve je vytvořeno prostředí simulace `env`. Následně je definována procesní funkce `robot`, která představuje aktivitu jednoho robota. Procesní funkci je v argumentech předáno simulační prostředí `env`, jméno robota `name` a směr `direction`, kterým robot bude chodit. Prostředí následně pomocí zavolání

env.process() zaregistrovalo procesní funkce `robot`, čímž došlo k vytvoření entit robotů. SimPy tak může řídit průběh jejích aktivit prostřednictvím událostí. Nakonec je simulace spuštěna zasláním zprávy `env.run()`. Výstup tohoto příkladu by vypadal následovně:

```
0: Robot A se začal pohybovat směrem vpravo
0: Robot B se začal pohybovat směrem vlevo
2: Robot A se zastavil a rozhlíží se
3: Robot B se zastavil a rozhlíží se
3: Robot A se začal pohybovat směrem vpravo
4: Robot B se začal pohybovat směrem vlevo
.
.
.
```

3.1.3 Prostředí simulace

Již víme, že simulační prostředí spravuje čas simulace, plánování a zpracování událostí, a poskytuje také prostředky pro postupné provádění simulace. Další funkce prostředí je obsluha spuštění simulace. Simulace v SimPy se spouští zasláním zprávy `env.run()`, kde případný argument této zprávy záleží na tom, kdy má simulace skončit. Simulace obvykle končí po vyčerpání všech událostí, nebo po uplynutí předem stanoveného času. Pokud chceme, aby simulace skončila po vyčerpání všech naplánovaných událostí, necháme zprávu `env.run()` bez argumentu. Speciálním případem takto spuštěné simulace může být nekonečná simulace, a to například když kód procesní funkce běží ve smyčce `while True:`, jako jsme si ukázali v příkladu s roboty. Tato simulace samovolně nikdy neskončí a musíme ji ukončit násilně. V případě, že chceme ukončit simulaci po uplynutí předem stanoveného času zašleme zprávu `env.run()` s argumentem `until=time`, kde `time` je hodnota datového typu `integer` nebo `float`, na které se má simulační čas zastavit, například `env.run(until=10)`. Simulace se zastaví v okamžiku, kdy čas dosáhne hodnoty 10, ale neprovede žádné události naplánované na čas 10. Poslední variantou je spuštění simulace do doby, než nenastane konkrétní událost `env.run(until=event)`.

3.1.4 Události

SimPy vytváří události prostřednictvím modulu `simpy.events`. Základní třídou pro všechny události je `simpy.events.Event`, která definuje společné vlastnosti a metody, jenž dědí všechny specializované typy událostí.

Z předešlých kapitol již víme, že `Timeout` je vestavěná událost, která je dokončena po uplynutí zadaného času. Používá se pro simulaci aktivity, která trvá určitou dobu. `Initialize` je událost, která signalizuje inicializaci komponenty a využívá se obvykle interně v SimPy. Komponentou je například objekt

`Resource`, který představuje přístup k omezenému zdroji. O nich se více dozvíme později. `Process` je událost reprezentující běh generátorové funkce. Dokončení této události znamená, že skončila nějaká aktivita. V některých situacích je potřeba čekat na více událostí současně, což SimPy umožňuje pomocí událostí, které obsahují seznam dalších událostí. Událost `AllOf` nastane ve chvíli, kdy nastanou všechny události v jejím seznamu. Událost `AnyOf` je naopak událost, která nastane už při první nastalé události ze seznamu.

Třída `Event` kromě vestavěných událostí umožňuje vytvořit vlastní události mající podobné vlastnosti jako vestavěné. Vlastní událost vznikne vytvořením nové instance třídy `Event`, které předáme prostředí, v němž se simulace odehrává. Vytvoření uživatelské události může vypadat například takto
`my_event = simpy.Event(env)`, kde `env` je prostředí. Uživatelsky definovaná událost je užitečná, když potřebujeme simulovat změnu stavu, na kterou SimPy nemá speciální událost. Typickými příklady jsou čekání na externí signál.

U uživatelsky definovaných událostí lze explicitně vyvolat jejich nastání. Jakmile událost nastane, může být vyhodnocena buď jako úspěšná, nebo jako neúspěšná. Úspěšné nastání události se vyvolá zasláním zprávy
`Event.succeed()`, která může volitelně nést návratovou hodnotu reprezentující výsledek dané aktivity. Tato hodnota je následně předána procesům, které na danou událost čekají.

V případě vzniku chybového stavu během zpracování procesní funkce lze událost vyhodnotit jako neúspěšnou zasláním zprávy `Event.fail()`. Tato metoda přijímá instanci výjimky, která je následně vyvolána v procesech čekajících na danou událost, čímž je umožněno standardní ošetření chybových stavů.

3.1.5 Sdílené zdroje

Sdílené zdroje představují objekty, o které soutěží více procesů současně. Pokud je zdroj obsazen nebo jeho kapacita neumožňuje další operaci, proces musí čekat ve frontě, dokud se zdroj neuvolní. SimPy rozlišuje tři typy sdílených zdrojů.
Resources jsou zdroje, které mohou být současně využívány libovolným omezeným počtem procesů. U objektů `resource` je sledováno počet současných uživatelů. V této práci využívám pouze zdroje tohoto typu, nicméně pro úplnost jsou zde uvedeny i další typy sdílených zdrojů, které SimPy nabízí. Podrobněji se však zaměříme pouze na zdroj typu `resource`.
Containers představují zdroje, ve kterých se pracuje s množstvím. Procesy do něj mohou dávat nebo brát množství hmoty.
Stores jsou zdroje, které uchovávají konkrétní objekty. Procesy mohou vložit nebo vybrat jednotlivý objekt, který má své specifické vlastnosti a pořadí.

Všechny zdroje sdílejí stejný princip, zdroj má kapacitu a fronty čekajících procesů. Každý sdílený zdroj obsahuje kapacitu, frontu pro vkládání `put_queue`, frontu pro vybírání `get_queue`, a metody `put()` a `get()` (u zdoje `resource` metody `request()` a `release()`). Metoda `put()` se používá pro vkládání množství nebo objektu do zdroje, pokud je zdroj plný, proces musí čekat ve frontě `put_queue`, dokud se místo neuvolní. Metoda `get()` se používá pro získávání

množství nebo objektu ze zdroje, pokud je zdroj prázdný, proces musí čekat ve frontě `get_queue`, dokud se požadovaná položka nevloží. Tyto metody nevykonávají akci okamžitě, ale vrací událost, která nastane ve chvíli, kdy je možné požadavek splnit. Proces se klíčovým slovem `yield` pozastaví a pokračuje po splnění požadavku.

Zdroje `Resources` mohou být využívány omezeným počtem procesů současně. Aby proces směl daný `resource` používat, musí si k němu vyžádat přístup, a to metodou `request()`. Po skončení proces musí zdroj `resource(res)` uvolnit metodou `release(res)`. Parametr `res` je zdroj, který ke kterému chceme přistoupit, nebo ho uvolnit. Volání `request()/release()` je tedy formou volání `put()/get()` u zdroje `resource`. Uvolnění proběhne vždy okamžitě.

Simpy implementuje `Resources` ve třech variantách. První variantou je základní zdroj `Resource`, jeho parametry jsou prostředí a kapacita, ta musí být vždy kladné celé číslo a ve výchozím nastavení je kapacita rovna jedné, tedy `Resource(env, capacity=1)`. Metoda `Resource.request()` vytváří `request token`, který reprezentuje konkrétní žádost procesu o daný zdroj. Tento token je uložen ve zdroji po celou dobu jeho využívání a je následně předán metodě `release()`, která na jeho základě zdroj uvolní. Uvolnění `Resource` lze provést jednoduchým způsobem, kdy pomocí `resource.request(req)` zažádáme o zdoj `req` a poté čekáme, až bude volný. Následně proběhne aktivita, a nakonec zdroj uvolníme pomocí `resource.release(req)`.

```
1 req = resource.request()
2 yield req
3 yield env.timeout(1)
4 resource.release(req)
```

Zdrojový kód 3: Přístup a uvolnění zdroje

Aby se předešlo situacím, kdy by zdroj zůstal obsazený například v důsledku přerušení procesu nebo vzniku chyby, nabízí SimPy bezpečnější řešení v podobě **context manageru**. Ten využívá klíčová slova `with` a `as` a zajišťuje automatické uvolnění zdroje po opuštění příslušného bloku kódu:

```
1 with resource.request() as req:
2    yield req
3    yield env.timeout(1)
```

Zdrojový kód 4: Přístup a uvolnění zdroje pomocí with

V tomto případě je `request token` automaticky zaregistrován v rámci bloku `with`. Jakmile je blok opuštěn, SimPy zajistí uvolnění zdroje bez nut-

nosti explicitního volání metody `release()`. Z vnitřního pohledu lze chování `context manageru` zjednodušeně přirovnat k následující struktuře:

```
1 try:
2    yield req
3    yield env.timeout(1)
4 finally:
5    resource.release(req)
```

Zdrojový kód 5: Přístup a uvolnění zdroje pomocí with interně

Díky tomuto přístupu je zaručeno, že zdroj bude vždy korektně uvolněn, a to i v případě přerušení procesu, nebo vzniku výjimky. V této práci využívám objekt `resource` v jeho základní podobě. Pro úplnost však níže představím i další dva typy objektu `resource`, které základní typ rozšiřují.

Druhou variantou sdíleného zdroje `Resources` je `PriorityResource`. Tato třída je podtřídou základní třídy `Resource` a umožňuje procesům při každé žádosti o zdroj specifikovat prioritu. Požadavky s vyšší prioritou získají přístup ke zdroji dříve než požadavky s nižší prioritou. Poslední variantou zdroje `Resources` je třída `PreemptiveResource`. Tato třída umožňuje odebrat zdroj procesu, který jej právě využívá. To je užitečné, pokud jsou nové požadavky natolik důležité, že pouhé přednostní zařazení do fronty nestačí a je nutné procesu, který využívá daný zdroj, tento zdroj odebrat.

3.2 Knihovna Tkinter

Grafické uživatelské rozhraní aplikace je vytvořeno kombinací knihoven `Tkinter` a `CustomTkinter`. `Tkinter` [5] je základní knihovna pro tvorbu uživatelského rozhraní jazyka Python, která poskytuje širokou sadu ovládacích prvků, neboli *widgetů*. Mezi hlavní ovládací prvky patří `Frame`, `Label`, `Entry`, `Button`, `Canvas` a `Text`. `Frame` představuje kontejnerový prvek, který slouží k logickému seskupování dalších widgetů. Umožňuje vytvářet přehledné oddělené části rozhraní. `Label` je widget určený pro zobrazování textu nebo obrázků. Slouží jako popisek, nadpis nebo informační prvek. `Entry` je jednořádkové textové pole určené pro zadávání hodnot uživatelem. Tlačítko `Button` je interaktivní prvek, který reaguje na kliknutí uživatele. Tkinter `Button` je funkční, ale vizuálně jednoduchý, což je hlavní důvod, proč jsou v této práci tlačítka vytvářena knihovnou `CustomTkinter` [6], která umožňuje například narozdíl od standardního `Tkinteru` zaoblené rohy tlačítek. `Text` je víceřádkové textové pole, které umožňuje zobrazování nebo editaci delšího textu. `Canvas` je jeden z nejdůležitějších widgetů `Tkinteru`, slouží jako interaktivní plocha a umožňuje kreslení grafických objektů, vkládání obrázků a detekuje události myši.

`Tkinter` je založen na událostním *event-driven* modelu. To znamená, že aplikace sama od sebe nevykonává žádné akce, ale čeká na události generované

uživatelem nebo systémem. Událostí může být například kliknutí myší, stisk klávesy nebo změna hodnoty widgetu. `Tkinter` tyto události zpracovává prostřednictvím hlavní událostní smyčky aplikace *event loop*, která se spouští metodou `mainloop()`. Po zavolání `mainloop()` Tkinter vytvoří nekonečnou smyčku, která neustále čeká na nové události, vyhodnocuje je a předává je příslušným widgetům. Tato smyčka zároveň zajišťuje překreslování okna, aktualizaci widgetů a celkovou interaktivitu aplikace. Každý widget v Tkinteru může generovat různé typy událostí. Na tyto události lze reagovat pomocí callback funkcí, které se přiřazují buď přímo parametrem `command`), nebo pomocí metody `bind()`, která umožňuje reagovat na konkrétní typ události, například `<Button-1>` nebo `<Motion>`.

Událostní model Tkinteru je klíčový zejména pro interaktivní části aplikace, jako je editor festivalového areálu. Widget `Canvas` zde slouží jako hlavní pracovní plocha, na které uživatel může umisťovat objekty, kreslit zóny a ty poté navzájem propojit. Každá interakce je zpracována právě prostřednictvím event loopu, který zajistí volání odpovídající callback funkce a následné překreslení plátna.

3.3 Knihovna Pillow

Knihovna `Pillow` [7] je rozšířením původní knihovny `PIL` a slouží k práci s rastrovými obrázky v jazyce Python. Poskytuje širokou škálu funkcí pro načítání, úpravu, konverzi a ukládání obrázků v různých formátech, jako jsou PNG, JPEG nebo GIF. V kontextu této aplikace je `Pillow` využívána pro načítání ikon objektů festivalových stánků. Aplikace využívá dvě verze ikon, a to barevnou a šedou variantu. Šedé ikony slouží k vizuálnímu odlišení stánků, které mají otvírací dobu a jsou v daný okamžik zavřené.

Základním prvkem knihovny je třída `Image`, která umožňuje otevření obrázku pomocí metody `Image.open()`. Načtený obrázek je možné dále upravovat a provádět jednoduché transformace, například změnu velikosti obrázku pomocí `resize()`. Pro zobrazení obrázků v grafickém rozhraní `Tkinteru` je nutné převést objekt `Image` do formátu kompatibilního s Tkinterem. K tomu slouží třída `ImageTk.PhotoImage`, která vytvoří objekt vhodný pro vložení do widgetů, například do `Label` nebo `Canvas`.

4 Technická dokumentace

Cílem této části je přiblížit způsob, jakým jsou jednotlivé komponenty aplikace implementovány, jak spolu komunikují a jakým způsobem je realizována samotná simulace. Dokumentace se zaměřuje na klíčové prvky aplikace, jako je reprezentace a chování návštěvníků, práce se simulačním časem a hlavní rozhodovací algoritmy.

4.1 Architektura aplikace

Aplikace je implementována jako samostatná desktopová aplikace v jazyce Python, která nevyžaduje připojení k internetu ani žádné externí služby. Celá aplikace běží lokálně a jednotlivé části aplikace spolu komunikují přímo prostřednictvím interních datových struktur. Architektura je rozdělena do několika logických vrstev. Základ tvoří model simulace, který je vytvořen v knihovně SimPy a zajišťuje běh diskrétní simulace, plánování událostí a práci se simulačním časem. Nad simulační vrstvou se nachází datová vrstva, která obsahuje definice všech entit, a to návštěvníků, objektů, kapel a konfiguračních parametrů.

Samostatnou část tvoří uživatelské rozhraní, implementované pomocí knihovny Tkinter. Ta zajišťuje vizuální zobrazení festivalového areálu, interaktivní ovládání simulace, vykreslování stavů objektů a zobrazování výpisů ze simulace. Uživatelské rozhraní je oddělené od simulační logiky a komunikuje s ní prostřednictvím kontroleru, který propojuje vizuální prvky s datovým modelem a simulačními procesy.

4.1.1 Struktura aplikace

Strukturu aplikace je zobrazena na obrázcích 1 a 2. Aplikace je rozdělena do tří hlavních adresářů a spouštěcího souboru `main.py`. Prvním z nich je adresář `/data`, který obsahuje dva podadresáře a množinu konfiguračních souborů ve formátu `json`. Mezi těmito soubory se nacházejí dva klíčové konfigurační soubory, do nichž uživatel nesmí zasahovat ručně. Jedná se o soubor `festival_settings.json`, v němž je uložen festivalový areál vytvořený uživatelem. Při každém uložení se tento soubor přepíše a vždy tak obsahuje aktuální podobu areálu včetně všech objektů, které se v něm nacházejí. Součástí tohoto souboru jsou také dva zásadní Python slovníky, které se vytvářejí při ukládání a slouží k orientaci návštěvníků po festivalové mapě. Jedná se o slovníky `ACTIONS_BY_LOCATIONS` a `ACTIONS_MOVING`, jejichž význam bude podrobně vysvětlen v pozdějších kapitolách.

Druhým důležitým konfiguračním souborem je `lineup.json`, který obsahuje aktuálně používaný časový harmonogram kapel. Ostatní konfigurační soubory v adresáři `data` mohou ovlivňovat chování simulace, avšak nelze je upravovat z aplikace. Uživatel je však může rozšiřovat doplněním dalších hodnot. Například soubory `names.json` a `surnames.json` obsahují seznam jmen

a příjmení, z nichž simulace vybírá jména při generování návštěvníků. Uživatel může tato data libovolně rozšířit, přičemž je nutné doplnit i mapu příjmení pro správný převod mezi mužským a ženským příjmením při generování rodin. Podobně lze rozšiřovat soubor `bands.json`, který obsahuje seznam kapel využívaný při generování kapel vystupujících na festivalu. Soubor `foods.json` obsahuje informace o jídlech a stáncích s občerstvením, zatímco `drinks.json` obsahuje data o nápojích a stáncích s nápoji.

Adresář `data` dále obsahuje dva podadresáře: `emojis` a `gui_editable`. První z nich obsahuje `png` ikony využívané při vizuálním zobrazení objektů na mapě festivalového areálu. Druhý podadresář obsahuje konfigurační `json` soubory, které lze upravovat přímo z aplikace prostřednictvím grafického uživatelského rozhraní.

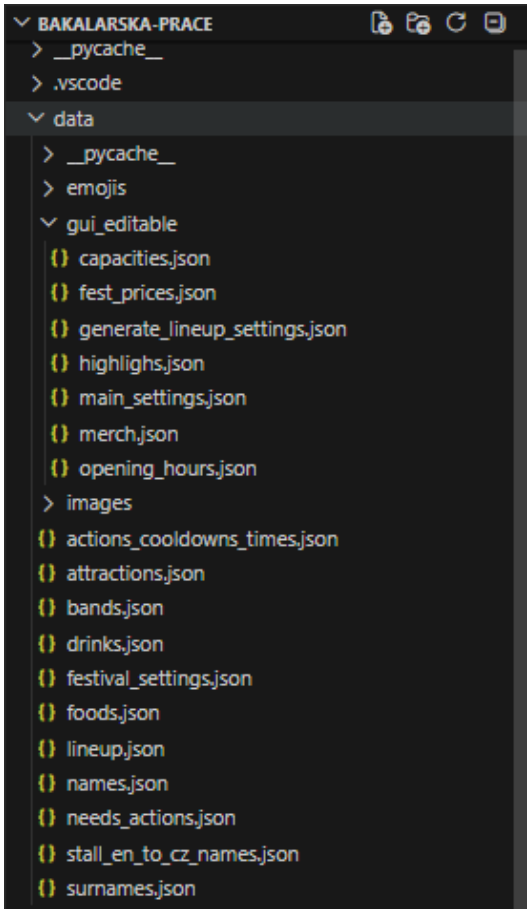
Druhým hlavním adresářem je `/outputs`, který obsahuje `json` soubory s výstupy ze simulace. Těmto výstupům se věnujeme v závěrečné části práce, konkrétně v kapitole 6 *Výsledky simulace*.

Třetím hlavním adresářem je `/src` obsahující zdrojové kódy. Jeho součástí je podadresář `/gui`, v němž se nachází soubor `gui.py`. Tento soubor představuje vstupní bod grafického rozhraní a zajišťuje inicializaci celé aplikace. Z `gui.py` jsou volány ostatní moduly, které se podílejí na načítání dat, ukládání konfigurací, validaci vstupů a následném spuštění simulace.

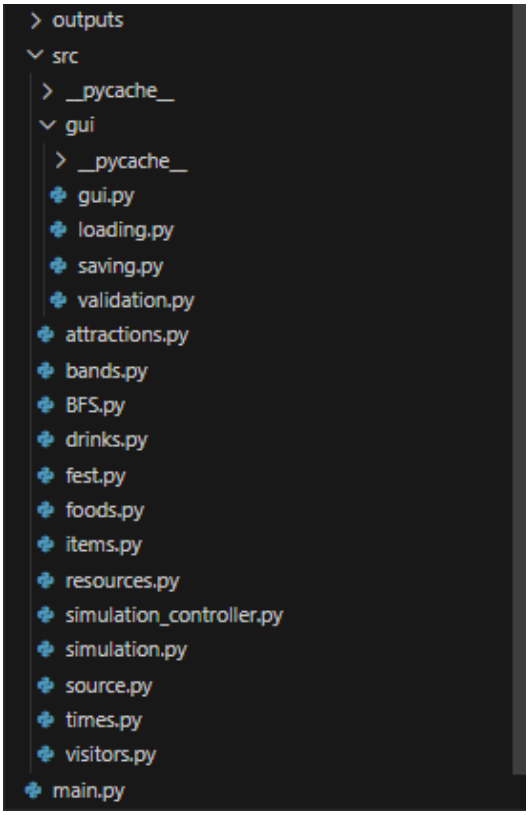
V adresáři `gui` se dále nacházejí soubory `loading.py` a `saving.py`, které obsahují implementaci logiky pro načítání a ukládání dat. Posledním souborem v adresáři `gui` je `validation.py`. Tento modul zajišťuje validaci vstupních formulářů během nastavování simulace, a to hlavně kontrolu správnosti datových typů zadaných hodnot a povolených rozsahů.

Součástí adresáře `/src` jsou moduly reprezentující jednotlivé entity simulace, například `visitors.py`, `resources.py`, `items.py`, `foods.py`, `drinks.py` nebo `attractions.py`. Tyto moduly definují datové struktury a objekty, které se v simulaci vyskytují, včetně jejich vlastností a chování. Další skupinu tvoří moduly zajišťující samotný běh simulace, jako `simulation.py`, `simulation_controller.py`, `times.py` nebo `BFS.py`. Tyto soubory implementují logiku plánování událostí, pohybu návštěvníků po areálu, vyhodnocování potřeb a interakcí s jednotlivými objekty.

Součástí adresáře `/src` je také modul `source.py`, který slouží jako centrální místo pro načítání konfiguračních dat a definici základních struktur využívaných v simulaci. Modul určuje kořenový adresář projektu a zpřístupňuje cesty ke všem konfiguračním souborům prostřednictvím připravených proměnných. Dále obsahuje několik výčtových typů `enum`, které reprezentují kategorie používané v simulaci, jako jsou typy skupin návštěvníků, lokace v areálu, stavy atrakcí či věkové skupiny. Součástí modulu jsou také překladové tabulky názvů stánků mezi anglickou a českou verzí a seznam stánků bez otevírací doby, které jsou dostupné po celou dobu simulace.



Obrázek 1: Struktura aplikace – část 1



Obrázek 2: Struktura aplikace – část 2

4.2 Simulační čas

Aby bylo možné převádět interní simulační čas na skutečný čas ve formátu `HH:MM`, využívá aplikace třídu `TimeConverter` z modulu `times.py` v adresáři `/src`. Tato třída při inicializaci obdrží počáteční čas simulace zadaný uživatelem a instanci prostředí SimPy `env`, které spravuje průběh simulačního času. Následně je volána metoda `set_start_time_to_minutes()`, jež převede počáteční čas zadaný ve formátu `HH:MM` na počet minut uplynulých od půlnoci pomocí metody `format_time_string_to_mins(time)`.

Během simulace je aktuální reálný čas získáván jako součet počátečního času simulace v minutách a hodnoty `env.now`, která představuje skutečný počet minut uplynulých od začátku simulace. Tento převod zajišťuje metoda `get_real_time(time=None)`, která výslednou hodnotu převádí do správného formátu metodou `format_time_minutes_to_hours(minutes)`.

Parametr `time=None` umožňuje převádět i libovolný konkrétní čas. Tato funkcionalita se využívá například pro získání času pro výpis času začátku koncertu následující kapely. Níže je uveden zdrojový kód třídy `TimeConverter`.

```
class TimeConverter:
   def __init__(self, start_time, env):
      self.start_time = start_time
      self.env = env

   def set_start_time_to_minutes(self):
      self.start_time = self.format_time_string_to_mins(self.
         start_time)

   def get_start_time(self):
      return self.start_time

   def format_time_minutes_to_hours(self, minutes):
      minutes = int(minutes)
      minutes = minutes % 1440
      hours = minutes // 60
      mins = minutes % 60
      return f"{hours:02d}:{mins:02d}"

   def format_time_string_to_mins(self, time):
      hours, minutes = map(int, time.split(":"))
      total_minutes = hours * 60 + minutes
      return total_minutes

   def get_real_time(self, time=None):
      if not time:
         time = int(self.env.now)
      else:
         time = int(time)

      total_minutes = time + self.start_time
      return self.format_time_minutes_to_hours(total_minutes)
```

Zdrojový kód 6: Implementace převodu simulačního času

4.3 Datové struktury

Datové struktury představují základní stavební prvky simulačního modelu. Nyní se zaměříme na tři hlavní třídy, a to na třídy návštěvníka `Visitor`, skupiny `Group` a stánku `Stalls`.

4.3.1 Stuktura návštěvníka

Třída `Visitor` reprezentuje jednotlivého návštěvníka festivalu a slouží jako datová struktura uchovávající návštěvníkovy atributy a metody obsluhující jednotlivé aktivity návštěvníka. Návštěvníci jsou generováni při přechodu do simulační části aplikace metodou `create_visitors(..)` v modulu `/src/visitors.py`, zatímco samotná implementace třídy `Visitor` je umístěna v modulu `/src/simulation.py`, jelikož veškerá logika interakcí návštěv-

níka s objekty během simulace je implementována v tomto modulu. Detailní popis generování návštěvníků a jiných entit, je uveden v následující kapitole.

Každý návštěvník má definovány základní identifikační údaje, těmi jsou návštěvníkovo ID, jméno, příjmení, pohlaví, věková kategorie a věk. Věková kategorie je reprezentována pomocí výčtového typu `Age_category`, který rozlišuje čtyři věkové kategorie návštěvníků, a to dítě, mladistvý, dospělý a senior. Kategorie se při generování návštěvníků přiřazuje náhodně dle typu skupiny, do které návštěvník patří. Samotný věk je následně generován náhodně v rámci rozsahu odpovídajícího dané věkové kategorii.

Dále jsou u návštěvníka definovány vlastnosti, jako například trpělivost, tendence utrácet peníze, frekvence hladu či tolerance alkoholu, a preference jako oblíbené jídlo, nápoj, alkoholický nápoj, zda kouří či seznam oblíbených kapel. Vlastnosti jsou vyjádřeny na škále 1–10, kde vyšší hodnota znamená silnější tendenci daného chování.

Součástí datové struktury je také aktuální stav návštěvníka, který je reprezentován sadou hodnot v rozsahu 0–100. Tyto hodnoty vyjadřují míru uspokojení jednotlivých fyziologických potřeb, jako je hlad, žízeň, energie, hygiena, potřeba toalety či nálada. Hodnota 100 znamená plné uspokojení potřeby, zatímco 0 představuje kritický stav. Výjimkou je opilost, kde 0 znamená střízlivost a 100 maximální opilost. Stavová část dále obsahuje informace o aktuální lokaci návštěvníka, množství peněz, dostupnosti vstupního náramku, čistotě rukou či zubů, volném čase a dalších parametrech ovlivňujících rozhodování.

Inventář návštěvníka obsahuje předměty, které má návštěvník u sebe. Patří sem stan, mobilní telefon, plastový kelímek, zakoupený merch, podpisy kapel z autogramiád a případně zakoupené jídlo či pití. Inventář dětí je omezený a nikdy neobsahuje telefon a stan, děti také nevlastní žádné peníze.

Třída `Visitor` dále uchovává informace o aktuální rychlosti pohybu, která je odvozena od základní rychlosti, tu má každý návštěvník generovanou v rámci rozsahu své věkové kategorie, a modifikována podle fyziologického stavu návštěvníka, zejména podle energie, nálady a stavu opilosti. Součástí struktury je také příznak, zda návštěvník právě vykonává aktivitu a seznam posledních provedených aktivit z důvodu časového omezení opakování stejné akce v krátkém časovém úseku. Důležitými atributy jsou `actual_goal` a `individual_need`. Atribut `individual_need` uchovává aktuální potřebu, kterou se návštěvník snaží uspokojit, zatímco atribut `actual_goal` představuje konkrétní aktivitu, kterou chce na základě této potřeby vykonat.

4.3.2 Struktura skupiny návštěvníků

Třída `Group` reprezentuje skupinové uskupení návštěvníků, kteří se v simulaci pohybují a rozhodují společně. Skupiny vznikají již při generování návštěvníků. Metoda `create_visitors(..)` ve skutečnosti vrací dva seznamy, seznam všech vytvořených návštěvníků a seznam skupin, do kterých jsou tito návštěvníci rozděleni. Seznam jednotlivých návštěvníků slouží pouze k následnému doplnění jejich oblíbených kapel z načteného harmonogramu kapel. Logika rozhodování

návštěvníků pracuje primárně se skupinami, jelikož chování návštěvníků je určeno typem skupiny, do které patří.

Skupiny mohou mít tři různé typy, a to *jednotlivec*, *rodina* nebo *skupina*. V implementaci jsou tyto typy reprezentovány výčtovým typem `source.Groups`, který obsahuje hodnoty `INDIVIDUAL`, `FAMILY` a `GROUP`. Jednotlivec je skupina o jednom členu, skupina obsahuje náhodně dva až šest členů, a rodina se skládá z jednoho až dvou dospělých, a jednoho až tří dětí. Jednotlivec může být jakýkoliv návštěvník věkové kategorie mladiství a starší. Skupina může být namíchaná různě, ovšem děti jsou povolené pouze u rodin.

Každá skupina uchovává seznam svých členů, identifikátor skupiny a informaci o typu skupiny. Součástí datové struktury je také skupinová rychlost pohybu, která je vypočítávána jako průměr aktuálních rychlostí všech členů skupiny, upravený o hustotu návštěvníků v zóně, ve které se skupina nachází.

Skupina dále uchovává informace o skupinovém cíli `group_goal`, cílové zóně `group_target_zone`, aktuální zóně, ve které se skupina nachází `group_actual_zone`, počtu členů, kteří se mají sejít v cílové zóně `members_needed`, a počtu členů, kteří již jsou v cílové zóně `members_arrived`. Tyto atributy slouží k synchronizaci skupiny a budou více vysvětleny později. Součástí struktury je také skupinový objekt `group_object`, se kterým skupina interaguje, typicky, když jde celá skupina na stejnou atrakci.

Třída obsahuje atributy potřebné pouze pro rodiny, jako je počet dětí nebo počet již obsloužených dětí u aktivit, které vyžadují asistenci rodičů.

4.3.3 Struktura objektů

Třída `Stall` reprezentuje jednotlivé objekty festivalového areálu. *Objekt* je jakýkoliv prvek umístěný ve festivalovém areálu, zatímco *stánek* je objekt, který obsluhuje návštěvníky, například stánky prodávající jídlo a nápoje. Oba pojmy jsou však reprezentovány třídou `Stall`, která obsahuje zdroj `simpy.Resource` nebo více zdrojů.

Objekty třídy `Stall` jsou vytvořeny metodou `create_resources(..)` v modulu `/src/resources.py` stejně jako skupiny návštěvníků při přechodu aplikace do simulační části aplikace.

Každý objekt má definovaný typ, id, id objektů potřebných pro zobrazení na canvasu, název, český název, zónu ve které se objekt nachází, zdroj typu `Resource` a případně otevírací dobu. Instance třídy `Stall` představuje základní entitu, se kterou návštěvníci interagují, vstupují do ní, čekají ve frontě, využívají její kapacitu nebo obsazují její pozice.

Jednoduché objekty, jako stánky s jídlem, nápoji nebo běžné atrakce, jsou reprezentovány jedním zdrojem `simpy.Resource` s definovanou kapacitou. Složitější stánky jsou tvořeny více zdroji. Toalety jsou reprezentovány seznamem objektů `Resource`, kde třetina kapacity tvoří pisoáry a dvě třetiny klasické toitoi budky, přičemž každý zdroj má kapacitu jedna. Stání u pódia je reprezentováno slovníkem se třemi zdroji, které odpovídají třem sektorům, těmi jsou stání

v blízkosti pódia, stání uprostřed a stání vzadu, přičemž každý sektor pojme třetinu návštěvníků. Autogramiádový stánek obsahuje tři zdroje, zdroj s kapacitou jedna pro kapelu, která má autogramiádu, zdroj s kapacitou pět pro návštěvníky u kapely, a zdroj pro návštěvníky ve frontě s kapacitou definovanou uživatelem mínus pět návštěvníků, kterým kapela právě dává autogram.

Každý stánek uchovává také informaci o počtu návštěvníků, kteří jej aktuálně využívají, a opočtu návštěvníků ve frontě. Tyto hodnoty jsou získávány přímo z objektů `simpy.Resource` pomocí atributů `count` a `queue`. Rozhodovací logika návštěvníků následně pracuje s těmito hodnotami při výběru stánku, typicky hledá stánek s nejkratší frontou.

Stánky mohou mít definovanou otevírací dobu, která je nastavena uživatelem v možnostech nastavení aplikace. Simulační prostředí otevírá a zavírá stánky v příslušných časech pomocí samostatných procesů, které mění stav stánku. Atribut `opend` značí, zda je stánek otevřen, či nikoliv. Otevírací doba ovlivňuje dostupnost akcí pro návštěvníky, některé stánky jsou dostupné neustále, jiné pouze v definovaných časových intervalech.

Některé objekty obsahují také seznam pozic, které reprezentují fyzická místa, například nabíjecí stanice nebo louku na stanování. V případě nabíjecích stanic jsou pozice modelovány jako seznam, kde první prvek udává počet volných pozic a další prvek obsahují prázdné seznamy `[]`. Nabíjecí stanice tak využívá kombinaci zdroje `Resource` pro obsluhu návštěvníka a seznamu pozic, do kterých návštěvníci ukládají své telefony. Telefon je instance třídy `Phone` definované v souboru `/src/items.py`. Návštěvník si do inventáře uloží ID nabíjecího stánku a pozici, na které je jeho telefon uložen.

Stanové městečko je oproti tomu implementováno odlišně. Ačkoli má vytvořený zdroj `Resource`, ve skutečnosti jej nevyužívá a pracuje pouze s pozicemi reprezentovanými slovníkem. Klíčem slovníku je ID stanu a hodnotou je příslušný zdroj `Resource` daného stanu, který obsahuje informaci o jeho kapacitě. Majitel stanu při stavbě vloží svůj stan do slovníku pozic dané louky a uloží si objekt této louky do své struktury `accommodation`. Ostatní členové skupiny, kteří sdílejí stejný stan, si následně uloží stejný objekt louky, který má u sebe uložený majitel stanu. Obsazenost stanového městečka je tak určena podle počtu položek ve slovníku pozic, zatímco informace o počtu lidí ve stanu se získává z příslušného zdroje `Resource`, protože stany jsou reprezentovány jako zdroje s náhodně generovanou kapacitou. Kapacita stanu závisí na typu skupiny, jednotlivec může mít stan s kapacitou jedna až dvě osoby, skupina dva až čtyři osoby a rodina má jeden stan s kapacitou odpovídající počtu všech členů rodiny.

Atrakce jsou reprezentovány jako speciální typ objektu `Stall`, který kromě základních atributů obsahuje také instanci třídy `Attraction` v atributu `stall.attraction`. Třída `Attraction` je definovaná v modulu `/src/attractions.py` a poskytuje kompletní logiku provozu atrakce, včetně řízení kapacity, stavu atrakce a průběhu jednotlivých jízd. Každá instance třídy `Attraction` má uložený objekt `Resource` pocházející z příslušného `Stall`, protože právě tento zdroj určuje maximální kapacitu atrakce a umožňuje sledovat

aktuální počet návštěvníků. Na základě těchto informací se atrakce rozhoduje, zda zahájí jízdu.

Atrakce zahájí jízdu, když je její kapacita plně obsazena, nebo když je naplněna alespoň z poloviny a zároveň uplynula polovina maximální čekací doby. Jízda se zahájí také v případě, že uplynula maximální čekací doba a na atrakci se nachází alespoň jeden návštěvník.

Třída obsahuje stav atrakce (`WAITING` nebo `RUNNING`), počítadlo aktuálních návštěvníků a dvojici událostí `ride_start` a `ride_end`, které slouží k synchronizaci návštěvníků čekajících na začátek nebo konec jízdy.

Metoda `run()` je spouštěna jako samostatný proces SimPy a neustále vyhodnocuje, zda jsou splněny podmínky pro zahájení jízdy. Jízda atrakce se zahájí událostí `ride_start` a stav atrakce se přepne na `RUNNING`. Po uplynutí definované délky jízdy nastane událost `ride_end`, která stav vrátí zpět na `WAITING`.

4.3.4 Struktura festivalu

Struktura festivalu je reprezentována třídou `Festival` definovanou v souboru `/src/fest.py`, která slouží jako centrální datový objekt uchovávající veškeré informace o simulovaném systému. Třída obsahuje seznam všech skupin návštěvníků, počet festivalových dní, kompletní program koncertů, seznam stánků rozdělených podle jednotlivých zón a také ceník vstupenek a služeb. Uchovává rovněž otevírací doby jednotlivých stánků a strukturu možných akcí, která se dynamicky mění podle toho, zda jsou otevřeny stánky uvnitř nebo vně festivalového areálu. Aktuální stav dostupných akcí je řízen atributem `actual_possible_actions`, jenž se automaticky aktualizuje procesem `watch_possible_actions()`.

Třída dále uchovává informace o právě hrající kapele, probíhající autogramiádě a pořadí kapel, které se budou podepisovat. Metody třídy poskytují přístup k cenám, programu, stánkům, atrakcím, aktuálnímu dni festivalu i k otevíracím dobám. `Festival` tak představuje datovou základnu simulace, ze které ostatní části systému, zejména `SimulationController`, čerpají informace potřebné pro řízení průběhu simulace.

4.3.5 Stuktura kontroleru

Kontroler je objekt třídy `SimulationController` a představuje hlavní řídicí komponentu simulace, která pracuje s daty uloženými ve třídě `Festival` a zajišťuje jejich průběžné vyhodnocování. Implementace třídy `SimulationController` se nachází v souboru `/src/simulation_controller.py`. Kontroler spravuje dynamická data o průběhu simulace. Nad těmito daty vytváří strukturu `simulation_state`, která slouží jako aktuální přehled o stavu celé simulace. Tato struktura obsahuje informace o aktuálním čase, počtu návštěvníků v jednotlivých zónách a detailní statistiky všech objektů, jako je počet obsloužených návštěvníků, počet lidí ve frontě, obsazenost stanového městečka, počet nabíjených telefonů nebo rozložení návštěvníků u pódia.

Při inicializaci kontroler vytvoří počáteční stav simulace metodou `create_simulation_state()`, která projde všechny zóny a stánky festivalu a uloží jejich základní parametry. Během běhu simulace je tento stav průběžně aktualizován metodou `get_actual_state()`, jež načítá aktuální hodnoty z objektů a zón, a zapisuje je do struktury `simulation_state`. Kontroler zároveň uchovává pomocné informace, například počet zobrazených logů, stav načtení festivalového areálu nebo mapu objektů podle jejich ID, která slouží pro rychlý přístup ke statistikám konkrétních objektů.

Kontroler řídí časový průběh simulace. Metoda `move_forward_by_time()` umožňuje krokovat prostředí SimPy po jednotlivých časových jednotkách až do dosažení požadovaného času, zatímco plynulý režim simulace lze zapnout nebo vypnout pomocí metod `start_smooth_simulation()` a `stop_smooth_simulation()`. Tyto funkce jsou využívány grafickým uživatelským rozhraním, které prostřednictvím kontroleru ovládá posun simulovaného času a načítání dat aktuálního stavu simulace.

4.4 Dostupné aktivity návštěvníků

Návštěvníci mohou v simulaci vykonávat širokou škálu aktivit, které odpovídají typickému chování na hudebním festivalu. Jednotlivé aktivity jsou mapovány pomocí slovníku `NEEDS_ACTIONS`, který převádí aktuální potřebu návštěvníka na konkrétní aktivitu. Dostupnost aktivit v konkrétní simulaci závisí na tom, jaké objekty jsou ve festivalovém areálu umístěny. Kompletní seznam aktivit je uveden níže.

Přesuny

- `GO_TO_NAZEV_ZÓNY` – Přesuny mezi zónami festivalu.
- `GO_TO_FESTIVAL_AREA_ENTRY` – Přesun do zóny `Festivalový areál`, u kterého je nutné projít vstupní kontrolou,

Fyziologické potřeby

- `GO_FOR_FOOD` – Nákup jídla,
- `GO_FOR_DRINK` – Nákup nápojů,
- `EAT` – Konzumace jídla,
- `DRINK` – Konzumace nápoje,
- `USE_TOILET` – Použití toalety,
- `SLEEP_IN_TENT` – Spánek ve stanu,

Hygiena

- `WASH` – Mytí rukou,
- `BRUSH_TEETH` – Čištění zubů,
- `USE_SHOWER` – Sprcha,

Kouření a relaxace

- `SMOKE` – Kouření cigaret,
- `GO_SMOKE_WATER_PIPE` – Kouření vodní dýmky,
- `BUY_CIGARS` – Nákup cigaret,
- `GO_CHILL` – Odpočinek v chill zóně,
- `SIT` – Sezení u stolů, je možné u stolu jíst a pít,

Program festivalu

- `ATTEND_CONCERT` – Účast na koncertu,
- `ATTEND_SIGNING_SESSION` – Účast na autogramiádě,
- `VISIT_ATTRACTION` – Návštěva atrakce,

Nákupy a interakce se stánky

- `BUY_MERCH` – Nákup upomínkového předmětu,
- `RETURN_CUP` – Vrácení kelímku,
- `WITHDRAW` – Výběr peněz z bankomatu,

Vstupní a ubytovací činnosti

- `BRACELET_EXCHANGE` – Výměna vstupního náramku, nákup festivalové vstupenky, případně vstupenky do stanového městečka,
- `PITCH_TENT` – Stavění stanu,

Telefon

- `CHARGE_PHONE` – Nabíjení telefonu,
- `TAKE_PHONE` – Vyzvednutí nabitého telefonu,

Ostatní aktivity

- `FOLLOW_PARENTS` – Následování rodičů dítětem,
- `DO_NOTHING` – Nedělání ničeho, pokud mají návštěvníci hodně volného času a mají všechny potřeby uspokojené, mohou se maximálně dvě hodiny jen volně procházet,
- `DEPARTURE` – Dočasný odchod z festivalu pro návštěvníky, kteří nejsou ubytováni ve stanu.

4.5 Simulace koncertů a autogramiád

Koncerty a autogramiády kapel zpracovává funkce `set_bands()` z modulu `/src/bands.py`, která pracuje s časovým harmonogramem kapel. Ten je reprezentován slovníkem a načítá se z konfiguračního souboru `/data/lineup.json`. Konkrétní podoba harmonogramu závisí na uživatelském nastavení popsaném v části 5.2.2 `Nastavení kapel` v uživatelské dokumentaci. Uživatel může lineup vytvořit ručně, nebo jej nechat vygenerovat. Generování kapel obstarávají funkce `create_lineup()` a `create_schedule()`. První funkce vybere pro každý den zadaný počet kapel, zatímco `create_schedule()` kapely seřadí podle atributu `popularity` a nastaví časy koncertů a autogramiád podle uživatelských parametrů.

Funkce `set_bands()` nejprve vypočítá pořadí autogramiád pomocí `get_order_of_signing_sessions()` a uloží jej do objektu `Festival`. Následně pro každou kapelu vytvoří dva paralelní SimPy procesy, a to `band_play()` pro průběh koncertu a `band_go_to_signing_session()` pro průběh autogramiády.

Proces `band_play()` nejprve čeká do okamžiku začátku koncertu kapely pomocí `env.timeout(..)`, poté získá zdroj pódia `stage.resource.request()` a čeká, než nastane čas konce koncertu. Po skončení koncertu proces vypočítá a zaloguje délku pauzy do následujícího vystoupení a uvolní zdroj. Proces `band_go_to_signing_session()` funguje obdobně, pracuje však se stánkem na autogramiády.

4.6 Logika návštěvníků

Tato podkapitola se věnuje nejdůležitější části simulace, a tou je logika chování návštěvníků.

Simulace je implementována v simulačním modulu `src/simulation.py`. Simulace začíná umístěním návštěvníků na festival do zóny `Spawn zóna`. Při přechodu uživatele do simulačního režimu se v modulu `src/gui/gui.py` spustí proces prostředí SimPy `env.process(simulation.spawn_groups(...))`. Tento proces postupně přidává jednotlivé skupiny návštěvníků na festival.

Funkce `spawn_groups` přijímá instanci kontroleru a seznam skupin návštěvníků. Z kontroleru získá objekt festivalu a z něj se načte časový harmonogram koncertů, vypočítá rozdíl mezi časem začátku simulace a časem začátku prvního koncertu. Tento časový úsek se následně vydělí počtem skupin návštěvníků, čímž vznikne interval, v němž budou jednotlivé skupiny přicházet na festival.

Pro každou skupinu funkce spustí samostatný proces `env.process(group.group_decision_making(controller))`. Tato funkce představuje hlavní řídicí mechanismus, který určuje chování skupiny návštěvníků v průběhu celé simulace. Než si jej vysvětlíme, musíme si napřed popsat metody, s kterými tato metoda pracuje. Při práci s členy skupiny používám označení `member`. Toto pojmenování vychází přímo z proměnné používané

ve `for-cyklu`, který postupně prochází všechny členy skupiny. Kdykoliv tedy funkce volá metody třídy `Visitor`, volá je právě na proměnné `member`.

4.6.1 Metoda urgent_need

Metoda třídy `Visitor` `urgent_need(..)` slouží k určení nejakutnější potřeby návštěvníka v daném okamžiku. Metoda při výběru potřeby zohledňuje systém časových blokací aktivit, který zajišťuje, že návštěvník nemůže vykonávat stejnou činnost příliš často za sebou. Každý návštěvník má vlastní proces `cooldown_actions(..)`, který v pravidelných intervalech kontroluje seznam posledních vykonaných aktivit uložený v atributu `last_actions`. Každá aktivita má definovaný časový limit, po jehož uplynutí je možné ji provést znovu. Proces proto u každé uložené aktivity ověří, zda její čas blokace již vypršel. Pokud ano, aktivita se ze seznamu odstraní, pokud ne, zůstává dále blokovaná. Podmínka validity potřeby se ověřuje metodou `is_need_valid(need)`, která převádí danou potřebu na odpovídající aktivitu pomocí slovníku `NEEDS_ACTIONS` uloženého v modulu `source`. Tento slovník obsahuje názvy potřeb jako klíče a názvy aktivit jako jejich přiřazené hodnoty.

Metoda postupně zohledňuje několik oblastí, mezi které patří základní festivalové požadavky, jako je získání vstupního náramku nebo postavení stanu, fyziologické potřeby, hygienické návyky, finanční situace, stav mobilního telefonu a preference návštěvníka. Metoda vrací řetězec reprezentující nejurgentnější potřebu návštěvníka, který se následně využívá v metodě `next_move(..)` při výběru konkrétní aktivity. Pokud vybraná potřeba splňuje podmínky validity, metoda ji okamžitě vrátí. V opačném případě vypočítá hodnoty důležitosti zbývajících potřeb a prochází potřeby od nejvyšší po nejnižší, a vrátí první potřebu, která je validní.

4.6.2 Metoda resolve_need

Metoda `resolve_need(..)` slouží k určení konkrétní aktivity, kterou má návštěvník vykonat na základě své aktuální potřeby. Jejím úkolem je převést potřebu na odpovídající akci, a to buď přímo v aktuální zóně, nebo nalezením nejkratší cesty do zóny, kde je možné danou potřebu splnit. Metoda nejprve načte seznam dostupných aktivit v jednotlivých zónách festivalu ze slovníku `ACTIONS_BY_LOCATIONS`, který je uložen v konfiguračním souboru `/data/festival_settings.json`. Tento slovník obsahuje pro každou zónu slovník ve tvaru `potřeba : akce` a je vytvořen při ukládání festivalového areálu.

Po získání aktuální zóny návštěvníka metoda nejprve ověří, zda je možné splnit danou potřebu přímo v této zóně. Pokud ano, vrátí metoda řetězec s názvem odpovídající aktivity. Pokud ne, vyhledá všechny zóny, které danou potřebu podporují, a vytvoří seznam potenciálních cílových zón. Pokud žádná zóna danou potřebu neumožňuje, metoda vrátí hodnotu `None`. Tato situace by však za

běžných okolností neměla nastat, protože metoda `urgent_need(..)` vybírá pouze takové potřeby, které je možné na festivalu v daném okamžiku splnit.

V případě, že lze splnit danou potřebu v jiné zóně, metoda využívá algoritmus *Breadth-First Search* k nalezení nejkratší cesty mezi aktuální zónou a některou z cílových zón. Z tohoto důvodu je metoda `resolve_need(..)` společně s implementací algoritmu definována v modulu `/src/BFS.py`. Algoritmus *Breadth-First Search* budeme označovat zkratkou BFS. Festival je v tomto kontextu reprezentován jako orientovaný graf, kde jednotlivé zóny představují vrcholy a přesuny mezi nimi tvoří hrany. Tyto přesuny jsou definovány ve slovníku `ACTIONS_MOVING`, který pro každou zónu obsahuje seznam možných přesunů ve tvaru `GO_TO_NÁZEV_ZÓNY`. Přesuny do zóny `FESTIVAL_AREA` mohou obsahovat i označení `ENTRY`. Toto označení znamená, že návštěvník musí vstoupit do festivalového areálu přes objekt vstupu.

Algoritmus BFS postupně prochází sousední zóny podle definovaných přesunů a vytváří cestu, která vede k nejbližší zóně schopné danou potřebu splnit. Výsledkem je první krok této cesty, tedy konkrétní přesun ve formě akce typu `"GO_TO..."`. Metoda tedy vrací název aktivity odpovídající návštěvníkově potřebě, nebo název aktivity přesunu do zóny, kde je možné danou potřebu splnit.

4.6.3 Metoda next_move

Metoda třídy `Visitor` `next_move(..)` představuje další krok v rozhodovací logice návštěvníka a propojuje dříve zmíněné metody.

Hlavní část metody probíhá v cyklu, který končí ve chvíli, kdy je určena konkrétní aktivita. Pokud návštěvník nemá v atributu `actual_goal` uloženou žádnou naplánovanou aktivitu, zavolá metodu `urgent_need(..)`, která určí jeho aktuální potřebu. Tuto potřebu si uloží do atributu `individual_need` a pomocí slovníku `NEEDS_ACTIONS` ji převede na odpovídající aktivitu, kterou si uloží jako nový `actual_goal`.

Pokud má návštěvník uložený cíl a tato aktivita je splnitelná v jeho aktuální zóně, metoda ji vrátí. Atributy `individual_need` a `actual_goal` se v tomto okamžiku nastaví na hodnotu `None`, protože potřeba byla úspěšně vyřešena. Pokud má návštěvník uložený cíl, ale tento cíl není možné splnit v jeho aktuální zóně, metoda zavolá `resolve_need(..)`, která určí vhodný přesun do zóny, kde lze danou aktivitu vykonat. V takovém případě metoda vrací aktivitu typu `GO_TO...`.

Atributy `individual_need` a `actual_goal` umožňují návštěvníkovi pokračovat v již zvolené aktivitě i v případě, že se musí přesunout do jiné zóny. Bez těchto atributů by návštěvník při každém průchodu rozhodovací logiky znovu vyhodnocoval své potřeby a mohl by tak měnit plánovanou aktivitu ještě předtím, než se dostane do zóny, kde ji lze splnit.

Skupiny typu `GROUP` metodu `next_move(..)` nevyužívají a pracují pouze s metodami `urgent_need(..)` a `resolve_need(..)`, protože skupinové chování je řízeno odlišným mechanismem. Z tohoto důvodu musí metoda `resolve_need(..)` samostatně ověřovat, zda je možné aktivitu splnit přímo

v aktuální zóně, i když tuto kontrolu již provádí metoda `next_move(..)` u jednotlivců a rodin.

4.6.4 Metoda group_decision_making

Metoda `group_decision_making` představuje hlavní řídicí mechanismus a jejím úkolem je řídit logiku chování jednotlivých typů skupin. Všechny tři typy skupin návštěvníků mají společný základ průběhu funkce. Manipulace se členy skupiny probíhá pomocí konstrukce `for member in self.members`. Díky čemuž je možné používat stejný mechanismus pro výpočet rychlosti přesunu a další skupinové operace, aniž by bylo nutné vytvářet speciální výjimky pro jednotlivce. Na začátku každého průchodu cyklem se všem členům aktualizuje jejich stav pomocí funkce `member.update_stats(..)`. Tato funkce sníží hodnoty jejich statistik, a to na základě náhodné hodnoty generované funkcí `random.uniform(..)` z modulu `random` a podle individuálních vlastností konkrétního návštěvníka. Po této aktualizaci se logika funkce začne lišit podle typu skupiny.

4.6.4.1 Jednotlivec

V případě skupiny typu `INDIVIDUAL` se u člena skupiny ověří, zda právě nevykonává žádnou aktivitu. Návštěvník vykonává aktivitu tehdy, pokud je jeho stav nastaven pomocí atributu `member.is_busy` na hodnotu `True`. Tento atribut mají všichni návštěvníci a jeho aktuální stav se kontroluje metodou `member.get_is_busy()`. Pokud návštěvník žádnou aktivitu nevykonává, vybere se jeho následující aktivita pomocí metody `member.next_move(..)`. Pokud následující aktivita návštěvníka bude přesun do jiné zóny, nastaví se příznak `in_zone` na `False`, v opačném případě na `True`. Tato informace se následně zohlední při výpočtu jeho rychlosti přesunu pomocí funkce `self.count_group_walking_speed(..)`.

Poté metoda `set_visitor_busy()` nastaví návštěvníkovi příznak zaneprázdněnosti, aby během vykonávání aktivity nemohl zahájit další aktivitu. Nakonec se vytvoří proces `self.env.process(self.start_action(..))`, který dále převezme obsluhu aktivity návštěvníka. Tento princip se používá i u zbylých dvou typů skupin, avšak v rozšířenější formě.

4.6.4.2 Rodina

U skupin typu `FAMILY` je průběh metody `group_decision_making(..)` podobný jako u jednotlivce, avšak rozšířený o koordinaci rodičů a dětí. Při prvním průchodu cyklem je jeden z rodičů vybrán jako primární (`family_leader`). Rodiče se získají pomocí výběru všech členů s věkovou kategorií `source.Age_category.ADULT` a z této množiny se náhodně vybere jeden rodič. Primární rodič v každém průchodu cyklu vybere následující aktivititu celé rodiny pomocí `family_leader.next_move(..)`. Stejně jako u ostatních typů skupin se však nejprve ověří, zda všichni členové rodiny nevykonávají

žádnou aktivitu. Pokud je alespoň jeden člen rodiny zaneprázdněný, funkce počká jednu simulovanou minutu pomocí `yield self.env.timeout(1)` a pokračuje až ve chvíli, kdy jsou všichni členové rodiny dostupní.

Po určení aktivity se nastaví příznak `in_zone` a vypočítá rychlost přesunu rodiny. Následně metoda `self.can_child_do_the_action(action)` vyhodnotí, zda jsou děti schopné danou aktivitu vykonat samostatně. Pokud ano, vykonají stejnou aktivitu jako primární rodič, pokud ne, děti pouze následují rodiče pomocí aktivity `FOLLOW_PARENTS`. Informace o tom, zda je potřeba asistence rodiče, se ukládá do proměnné `child_asistance`.

Následně se všichni členové rodiny označí jako zaneprázdnění pomocí `member.set_visitor_busy()` a pro každého člena rodiny se spustí proces `self.env.process(self.start_action(..))`, který zajistí vykonání aktivity. U rodičů se rozlišuje, zda je rodič primární, tedy ten, který aktivitu vybral. Pokud ano a je potřeba asistence dětem, spustí se metoda `start_start_action(..)` s parametrem `child_asistance`. Druhý rodič vykoná aktivitu samostatně.

Pokud je potřeba asistence rodiče, aktivita rodiče se stává synchronizačním bodem celé rodiny. Rodič, který aktivitu vykonává, vytvoří uživatelsky definovanou událost `self.action_done = self.env.event()`, která reprezentuje okamžik dokončení rodičovské aktivity. Skupiny, tedy instance třídy `Group`, obsahují atribut `self.result_for_children`, který slouží k předání výsledku rodičovské aktivity dětem. Děti, které aktivitu samostatně vykonat nemohou, čekají na dokončení rodičovské aktivity čekáním na nastání události `yield self.action_done`. Jakmile rodič aktivitu dokončí, uloží výsledek do `result_for_children` a událost se ukončí zasláním zprávy `self.action_done.succeed()`. Tím se všechny čekající dětské procesy probudí a pokračují v aktivitě `FOLLOW_PARENTS`, a to tak, že si z atributu `result_for_children` převezmou výsledek rodičové aktivity, například obdrží nápoj nebo jídlo, které rodič zakoupil.

U aktivit výběru pěněz z bankomatu a vložení či vyzvednutí telefonu z nabíjecího stánku druhý rodič pouze doprovází primárního rodiče, jelikož tyto aktivity vždy může vykonat pouze jeden návštěvník. U těchto aktivit tedy druhý rodič pouze čeká na dokončení aktivitiy primárního rodiče. Pokud tyto aktivity potřebuje vykonat druhý rodič, může se od rodiny odpojit, aktivitu samostatně vykonat a následně se ke skupině opět připojit. Může nastat situace, kdy se po odpojení jednoho rodiče zbytek rodiny přesune do jiné zóny, zatímco odpojený rodič se stále nachází ve své původní lokaci. Z tohoto důvodu simulace při každém průchodu řídicí smyčky po ověření zaneprázdněnosti členů skupiny kontroluje, zda se všichni členové nacházejí ve stejné zóně. Pokud je některý člen rodiny lokalizován v odlišné zóně, je mu spuštěn samostatný proces přesunu do správné zóny, zatímco ostatní členové rodiny vyčkávají na jeho příchod.

4.6.4.3 Skupina

V případě skupiny typu `GROUP` je metoda `group_decision_making(..)`

nejkomplexnější částí rozhodovací logiky. Skupina představuje seskupení návštěvníků, kteří se snaží vykonávat aktivity společně, avšak každý člen má vlastní individuální potřeby. Z tohoto důvodu se členové skupiny mohou dočasně odpojovat a opět připojovat podle toho, zda jejich osobní potřeby kolidují se skupinovým cílem.

Na začátku každého průchodu se členové rozdělí do dvou režimů. Návštěvníci mají ve slovníku reprezentujícím jejich stav atribut `group_mode`, který určuje, zda se aktuálně řídí skupinovým chováním, nebo jednají samostatně. Členové se podle této hodnoty rozdělí do seznamů `group_mode_members` a `solo_mode_members`.

Skupina může rozhodovat o dalším kroku pouze tehdy, když všichni členové ve skupinovém režimu nevykonávají žádnou aktivitu. Pokud je alespoň jeden z nich zaneprázdněný, skupina počká jednu simulovanou minutu a pokračuje až ve chvíli, kdy jsou všichni členové připraveni.

Pokud skupina nemá nastavený skupinový cíl `group.actual_goal`, musí jej určit. Každý člen ve skupinovém režimu proto nejprve vyhodnotí svou individuální potřebu pomocí metody `urgent_need(..)`. Tyto potřeby se následně spočítají a skupina vybere tu, která se vyskytuje nejčastěji. Některé potřeby, jako například nabíjení telefonu, nejsou vhodné pro skupinové vykonávání, a proto jsou vyloučeny. Výsledná skupinová potřeba se uloží do atributu `group_need` a pomocí slovníku `NEEDS_ACTIONS` se převede na konkrétní aktivitu, která se uloží jako `group_goal`.

Dalším krokem je určení cílové zóny, kde lze skupinovou aktivitu vykonat. K tomu se využívá metoda `resolve_need(..)`, která vrátí buď přímo aktivitu, nebo akci typu `"GO_TO..."` představující přesun do jiné zóny. Pokud je vrácena akce přesunu, skupina musí zjistit celou cestu do cílové zóny. To se provádí opakovaným voláním `resolve_need(..)` v cyklu `while` s aktualizovanou zónou, dokud není nalezena poslední zóna, kde lze aktivitu vykonat. Výsledkem je atribut `group_target_zone`, který určuje, kam se má skupina přesunout.

Pokud má člen skupiny individuální potřebu s vysokou prioritou a tato potřeba není shodná se skupinovou potřebou, člen se přepne do individuálního režimu. Mezi tyto potřeby patří například toaleta, žízeň, hlad, nízká hodnota peněz, hygiena, kouření nebo nízká energie.

Pokud skupina má členy ve skupinovém režimu, následuje přesun do cílové zóny. Členové se rozdělí na ty, kteří již v cílové zóně jsou, a na ty, kteří se tam teprve musí přesunout. Pokud existují členové mimo cílovou zónu, skupina musí zajistit, aby členové v cílové zóně aktivitu nezačali vykonávat dříve, než dorazí zbytek skupiny. K tomu se využívá uživatelsky definovaná událost `self.group_is_ready = self.env.event()`, která slouží jako synchronizační bod.

Členové, kteří již v cílové zóně jsou, spustí proces `wait_for_complete_group()`, ve kterém čekají na dokončení události pomocí `yield self.group_is_ready`. Členové mimo cílovou zónu se mezitím

přesouvají pomocí funkce `find_the_way(..)`, která přijímá jako argumenty aktuální zónu návštěvníka a cílovou skupinovou zónu a vrací výsledek BFS algoritmu. Ten určí nejbližší krok směrem k cílové zóně. Jakmile člen dorazí do cílové zóny, skupina zvýší interní čítač `members_arrived`. Pokud je počet členů v cílové zóně roven počtu členů ve skupinovém režimu, událost se dokončí voláním `self.group_is_ready.succeed()`. Tím se všichni čekající členové probudí a skupina může pokračovat společnou aktivitou.

Rychlost přesunu vypočítá funkce `count_group_walking_speed(..)`, která zohledňuje, zda se člen přesouvá sám nebo společně se skupinou. Jakmile jsou všichni členové skupiny v cílové zóně, skupina vykoná společnou aktivitu. Každý člen se označí jako zaneprázdněn a spustí proces `start_action(..)`, který zajistí vykonání aktivity. Po dokončení aktivity se skupinový cíl nastaví na hodnotu `None` a skupina je připravena vybrat nový společný cíl.

Členové, kteří se odpojili kvůli individuální potřebě, se chovají jako jednotlivci. Po dokončení vlastní aktivity se těmto členům nastaví atribut `group_mode` opět na `True`.

4.7 Metoda start_action

Zatímco předchozí metody rozhodovací logiky určují, jakou aktivitu má návštěvník vykonat, metoda `start_action(..)` zajišťuje samotnou realizaci této aktivity. Jejím úkolem je vyhledat potřebný objekt festivalu, připravit všechny parametry aktivity a následně spustit odpovídající obslužnou metodu.

Metoda nejprve uloží čas přesunu návštěvníka do proměnné `travel_time`. U jednotlivců a rodin se používá skupinová rychlost chůze, zatímco u skupin typu `GROUP` se rozlišuje, zda se člen přesouvá společně se skupinou, nebo individuálně. Podle toho se použije buď `group_walking_speed`, nebo individuální rychlost uložená ve slovníku `individuals_walking_speed`, který má jako klíče návštěvníky v sólo režimu, a jako hodnoty jejich individuální čas přesunu.

Následně metoda vyhodnotí typ aktivity a podle jejího druhu vyhledá potřebný objekt festivalu. U přesunů se volají metody typu `go_to(..)`, které návštěvníka přesunou do cílové zóny. U aktivit vyžadujících interakci s objektem metoda nejprve vyhledá vhodný objekt pomocí funkcí `choose_stall(..)` nebo `find_stalls_in_zone(..)`. Pokud je potřeba vybrat konkrétní položku, například jídlo nebo pití, metoda zavolá funkce `choose_food(..)` nebo `choose_drink(..)`, které zohledňují preference návštěvníka, dostupnost položek a jeho finanční možnosti.

Každá aktivita má vlastní obslužnou metodu, která provádí její průběh. Metoda `start_action(..)` tedy pouze připraví všechny potřebné parametry a následně spustí proces SimPy pomocí `self.env.process(..)`, který zajistí časový průběh aktivity.

Metoda podporuje také speciální režimy, například asistenci rodiče dětem. Pokud rodič vykonává aktivitu za dítě, metoda předá informace o počtu dětí

a po dokončení aktivity vyvolá událost `action_done`, aby se dětské procesy mohly probudit a převzít výsledek aktivity. U skupin typu `GROUP` metoda navíc kontroluje, zda člen dorazil do cílové zóny, a pokud ano, aktualizuje interní čítač skupiny.

Po dokončení aktivity každý návštěvník zavolá `member.set_visitor_not_busy()`, čímž se mu nastaví atribut `is_busy` na `False`. Metoda `group_decision_making` poté může skupině vybrat novou aktivitu.

4.8 Vizualizace

Vytváření festivalového areálu a následná vizualizace průběhu simulace je postavena na widgetu `Canvas` z knihovny `Tkinter`, nad kterým je implementován editor festivalového areálu. Tento editor umožňuje uživateli vytvářet zóny, umisťovat objekty, upravovat jejich polohu, měnit velikost zón a vytvářet spojení mezi jednotlivými zónami festivalu. Celé GUI pracuje nad sadou globálních stavových proměnných, které uchovávají informaci o aktuálně vybraném objektu, zóně, režimu editoru a souřadnicích posledního kliknutí.

Editor reaguje na tři základní události myši, a to kliknutí, tažení a uvolnění tlačítka. Funkce `on_click` určuje, jaký prvek byl kliknut, a podle aktuálního režimu volá odpovídající obslužnou funkci. Funkce `on_drag` zajišťuje kreslení zón, přesouvání objektů nebo změnu velikosti zón. Funkce `on_release` ukončuje kreslení nebo přesun a ukládá výsledné souřadnice.

Editor funguje ve čtyřech režimech, které určují, jakým způsobem se zpracovává kliknutí a tažení myší. Režim `add` slouží k vytváření nových zón a umisťování objektů. Pokud uživatel klikne na plátno a má vybraný objekt, editor jej umístí do aktuálně vybrané zóny. Pokud není vybraný objekt, ale je vybrán typ zóny, kliknutí zahájí kreslení nové zóny uložením počátečního bodu a aktivací příznaku kreslení. Tažení myší pak aktualizuje velikost obdélníku reprezentujícího zónu. Editor zároveň kontroluje, zda zóna nebo objekt mohou být vytvořeny pouze jednou. Maximálně jednou může být vytvořena každá festivalová zóna a objekty `Podium` a `Stan na autogramiády`.

V režimu `edit` editor umožňuje výběr objektů, zón nebo spojovacích čar a jejich následnou úpravu. U objektů lze provádět jejich přesun v rámci zóny. Editor zároveň hlídá, zda objekt nepřesáhl hranice zóny. U zón lze tažením myši měnit jejich velikost. V tomto režimu je také možné mazat vybrané prvky pomocí klávesy Delete. Funkce `on_drag` podle typu vybraného prvku provádí jeho přesun nebo úpravu rozměrů a aktualizuje souřadnice uložené v datové struktuře. Konfigurace festivalového areálu je uchovávána v datové struktuře `zones_data`, která obsahuje souřadnice zón, seznam objektů v každé zóně a seznam spojovacích čar.

Režim `connect` slouží k vytváření spojení mezi zónami nebo mezi zónou a objektem vstupu. První kliknutí určí počáteční prvek, druhé kliknutí určí cílový prvek. Editor následně vypočítá vhodné souřadnice spojení, vykreslí čáru na

plátně a uloží spojení do obou zón v datové struktuře. Tato spojení se později využívají při kontrole dosažitelnosti zón pomocí algoritmu BFS před spuštěním simulace.

Režim `inspect` neslouží k úpravám, ale k zobrazení stavu simulace. Kliknutí na objekt zvýrazní jeho grafickou reprezentaci a editor načte aktuální stav objektu ze simulace. V pravém panelu se zobrazí informace o kapacitě, počtu obsluhovaných návštěvníků, frontě, otevírací době nebo například o aktuálně hrající kapele na pódiu. Tento režim umožňuje uživateli sledovat průběh simulace v simulačním režimu.

4.8.1 Vykreslování dat během simulace

Simulace může být spouštěna buď v rychlém nebo plynulém řežimu. Režimy se liší způsobem posunu času a jakým způsobem jsou aktualizována data v grafickém rozhraní.

Základní funkcí pro posun simulace je metoda kontroleru `controller.move_forward_by_time(time)`, která provede posun simulačního času o počet minut předaných jako argument metody. Metoda ve `while` cyklu opakovaně volá `self.festival_env.run(until=self.festival_env.now + 1)`, dokud není dosažen cílový čas nebo konec simulace. Pokud čas simulace je větší neč čas konce simulace, metoda vrátí hodnotu `True` a následuje ukončení simulace metodou `end_of_simulation()`.

Pro plynulý běh simulace slouží funkce `start_smooth_simulation()`, která aktivuje automatický režim kontroleru (`auto_mode = True`) a spustí v samostatném vlákně funkci `smooth_loop()`. V této funkci běží nekonečný `while` cyklus, který opakovaně posouvá simulaci o jednu minutu simulovaného času pomocí `move_forward_by_time(1)` a následně volá metodu `move_forward_actions()`, která z kontroleru načte aktuální data o stavu simulace metodou `controller.get_actual_state()`. Metoda nejprve aktualizuje čas simulace a počty návštěvníků v jednotlivých zónách. Následně projde všechny objekty festivalu a pro každý z nich aktualizuje statistiky uložené ve struktuře `simulation_state`. Tato data předá metodě `view_changes()`, která porovná předchozí stav s aktuálním stavem a provede všechny potřebné změny na plátně. To zahrnuje změnu barvy nebo ikony objektů podle jejich aktuálního stavu. Barvy objektů se mění podle předem definovaných prahových hodnot, které jsou načteny z konfiguračního souboru. Metoda `view_changes()` také aktualizuje údaj o aktuálním čase a dnu v simulaci zobrazený v horní části levého panelu simulačního režimu. Funkce `update_day_time_labels()` přepočítá aktuální den festivalu a reálný čas simulace a zobrazí je uživateli. Posledním krokem cyklu je uspání smyčky na jednu vteřinu. Plynulý režim tak umožňuje sledovat simulaci v reálném čase, přičemž každá sekunda reálného času odpovídá jedné minutě simulovaného času. Funkce `stop_smooth_simulation()` umožňuje plynulý režim kdykoli pozastavit.

Rychlý běh simulace provádí funkce `automatic_simulation()`, při kterém jsou všechny ovládací prvky aplikace dočasně deaktivovány a uživateli se zobrazí informační okno o probíhající simulaci. Simulace je následně vykonávána na pozadí podobně jako u plynulé simulace ve `while` cyklu po minutách simulačního času pomocí volání `env.run(until=env.now + 1)`, tentokrát se však cyklus se ukončí až po konci celé simulace. V tomto režimu se nevolá `move_forward_actions()`, a není tak aktualizováno uživatelské rozhranní. Simulace proběhne zrychleně bez možnosti interakce uživatele a možnosti sledovat průběh simulace přímo v aplikaci. Po dosažení konce festivalu jsou ovládací prvky opět aktivovány a simulace je ukončena metodou `end_of_simulation()`.

4.9 Inicializace simulace

Inicializace všech komponent potřebných pro běh simulace se provede po dokončení konfigurace festivalového areálu v editoru funkcí `start()` v modulu `/src/gui/gui.py`. Funkce se spustí po kliknutí uživatele na tlačítko `Start` a představuje přechod z editačního režimu aplikace do simulačního režimu. Na začátku funkce se nejprve ověří, zda byl festivalový areál načten nebo vytvořen a uložen, pokud ne, aplikace zobrazí chybové hlášení a simulace se nespustí. Následně se aktuální konfigurace areálu uloží do souboru `/data/festival_settings.json`.

Dalším krokem je zjištění, jaké aktivity jsou v dané konfiguraci festivalu možné. Funkce `get_possible_actions()` načte nastavení zón a jejich aktivit, rozdělí je podle dostupnosti ve festivalovém areálu nebo mimo něj a vytvoří čtyři množiny dostupných aktivit. Ty jsou následně uloženy do slovníku `possible_actions`, který se předává třídě `Festival`. Následuje kontrola konfigurace pomocí funkce `check_actions_and_connections(..)`, která ověřuje, zda jsou všechny zóny navzájem dosažitelné pomocí algoritmu BFS, a zda jsou v areálu umístěny všechny povinné objekty. Pokud kontrola zjistí chyby, funkce `start()` zobrazí uživateli seznam chyb a simulace se nespustí.

Po úspěšné validaci se editor přepne do režimu `inspect`, který slouží k vizuálnímu sledování průběhu simulace. Všechny editační prvky uživatelského rozhraní funkce nahradí ovládacími prvky simulace. Dále funkce zobrazí panel se statistikou počtu návštěvníků v zónách a textová pole `stall_log_box` a `messages_log_box` pro zobrazování výpisů ze simulace a informací o zvoleném objektu.

Následně funkce `start()` načte data ze všech konfiguračních souborů, které uživatel nastavil během přípravy simulace. Jedná se zejména o kapacity stánků, ceny služeb, nastavení upomínkových předmětů, hlavní parametry simulace, otevírací doby stánků a harmonogram kapel. Funkce poté vytvoří prostředí `SimPy` `festival_env` a časový konvertor `time_converter`. Funkce `create_resources(..)` v modulu `/src/resources.py` následně vytvoří všechny objekty festivalu.

Dalším krokem je zjištění, jaké konkrétní typy jídel a nápojů jsou na fes-

tivalu dostupné. Funkce `find_all_type_stall_at_festival(..)` najde všechny stánky prodávající jídlo a nápoje, a z nich vytvoří seznam dostupných jídel, nealkoholických nápojů a alkoholických nápojů. Pokud není k dispozici žádný stánek s nealkoholickými nápoji, aplikace zobrazí chybové hlášení a simulace se nespustí, protože by nebylo možné uspokojit základní potřebu žízně u návštěvníků nepijících alkohol.

Kapacita stanového městečka se vypočítá podle počtu luk pro stanování a jejich kapacit. Pokud je v konfiguraci povolena aktivita `energy`, tedy spánek ve stanu, vypočítá se maximální počet stanů, které mohou být na festivalu postaveny. Funkce následně vygeneruje návštěvníky a jejich skupiny pomocí funkce `create_visitors(..)` v modulu `/src/visitors.py`. Poté funkce načte časový harmonogram kapel a každému návštěvníkovi náhodně přiřadí oblíbené kapely pomocí funkce
`add_favorite_bands_to_visitor(..)`.

Na základě všech těchto dat funkce vytvoří instance třídy `Festival`, která obsahuje informace o prostředí simulace, skupinách návštěvníků, počtu dní, programu, objektech, cenách, možných aktivitách a otevíracích dobách stánků. Festival spustí proces `watch_possible_actions()`, který průběžně sleduje, jaké aktivity jsou v daném čase a konfiguraci dostupné. Pokud je v konfiguraci povolena aktivita `want_merch`, vytvoří se merch podle nastavených kapel a přiřadí se festivalu. Dále funkce nastaví funkcí `set_bands(..)` z modulu `/src/bands.py` koncerty kapel a případně autogramiády.

Pro řízení simulace funkce vytvoří instanci třídy `SimulationController` a nakonec spustí proces `spawn_groups(..)`, který postupně přidává skupiny návštěvníků na festival. Po dokončení inicializace se GUI přepne do simulačního rozhraní.

5 Uživatelská dokumentace

V této kapitole si ukážeme, jak simulační aplikaci zprovoznit a spustit. Následně se zaměříme na popis ovládacích prvků aplikace a uživatelského rozhraní.

5.1 Zprovoznění aplikace

Aplikace byla vyvíjena a testována na operačním systému Windows 11. Z tohoto důvodu je doporučeno aplikaci spouštět na platformě Windows. Spuštění aplikace je realizováno prostřednictvím skriptu `main.py`, který představuje hlavní vstupní bod celé aplikace. Pro správnou funkčnost je nutné mít nainstalovaný interpret jazyka Python (doporučena verze 3.10 nebo novější) a externí knihovny `SimPy`, `CustomTkinter` a `Pillow`.
Instalaci těchto knihoven lze provést v příkazovém řádku pomocí nástroje `pip` následujícími příkazy:

```
py -m pip install simpy
py -m pip install customtkinter
py -m pip install pillow
```

Knihovna CustomTkinter využívá standardní knihovnu `Tkinter`, která je součástí instalace jazyka Python.

Aplikaci je možné spustit v libovolném vývojovém prostředí podporujícím jazyk Python, například Visual Studio Code, případně přímo z příkazové řádky příkazem `py main.py`.

5.2 Uživatelské rozhraní

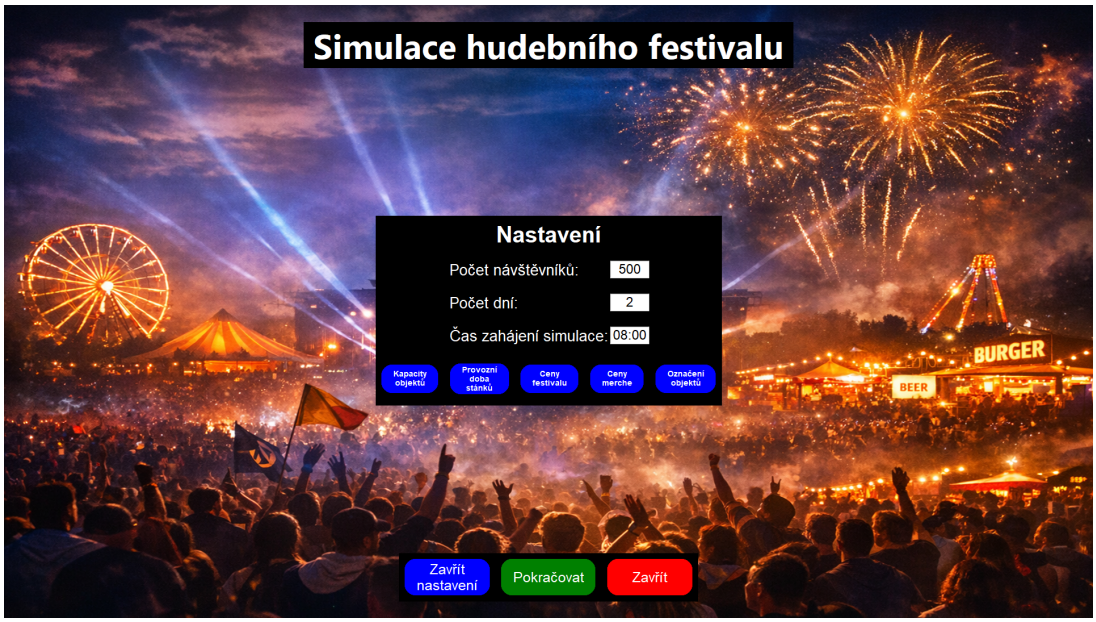
Úvodní obrazovka aplikace obsahuje spodní panel se třemi základními ovládacími prvky, kterými jsou tlačítka `Nastavení`, `Pokračovat` a `Zavřít`. Tlačítko `Nastavení` otevře nabídku nastavení parametrů simulace. Pomocí tlačítka `Pokračovat` se uživatel přesune k dalšímu kroku, ve kterém se nastavuje festivalový harmonogram kapel. Tlačítko `Zavřít` slouží k ukončení aplikace.

5.2.1 Nastavení simulace

Po kliknutí na tlačítko `Nastavení` se otevře základní konfigurační okno, ve kterém může uživatel definovat hlavní parametry simulace. Toto okno je zobrazeno na obrázku 3. Uživatel zde nastavuje tři klíčové údaje, a to celkový počet návštěvníků, počet festivalových dní a čas zahájení simulace. Všechny časové hodnoty v nastavení je nutné zadávat ve formátu HH:MM. Čas zahájení představuje okamžik, ve kterém simulace začíná, tedy kolik hodin bude na festivalu v momentě spuštění simulace. Na základě tohoto údaje se následně vypočítá celková doba trvání simulace, která je určena počtem festivalových dní, přičemž jeden den odpovídá 1440 minutám. Simulace tedy začíná v definovaném čase a končí ve stejnou denní hodinu po uplynutí zadaného počtu dní. Hodnoty těchto základních

parametrů se ověří a uloží ve chvíli, kdy uživatel klikne na tlačítko `Pokračovat` a přesune se k dalšímu kroku konfigurace.

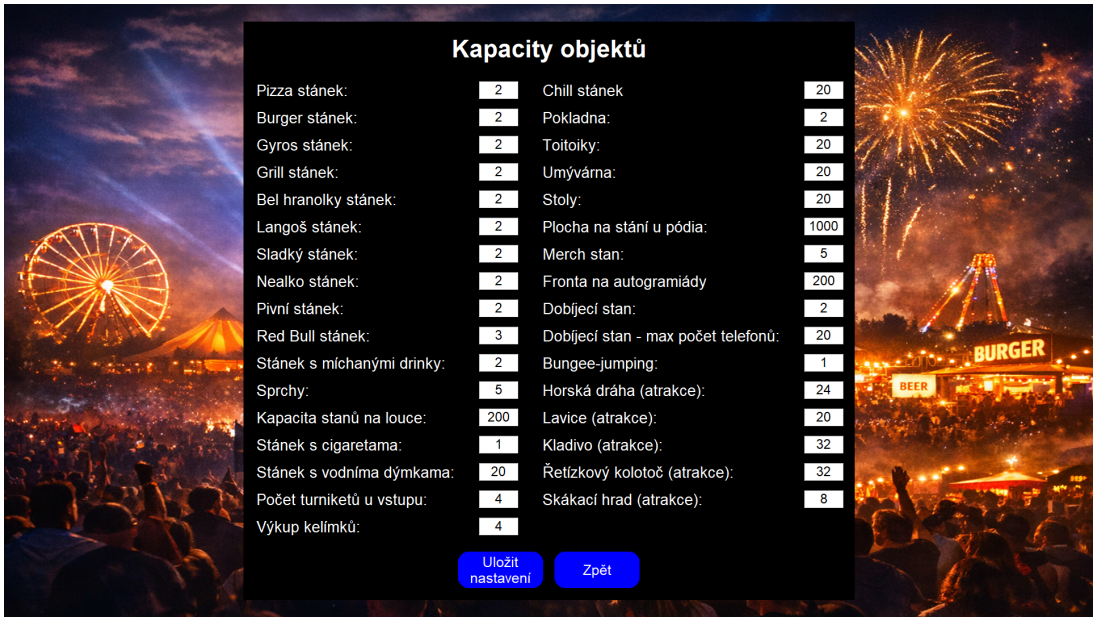
Ve spodní části okna se nachází tlačítka umožňující přístup k rozšířeným možnostem nastavení, které uživateli poskytují detailnější konfiguraci jednotlivých aspektů simulace.



Obrázek 3: Základní obrazovka nastavení

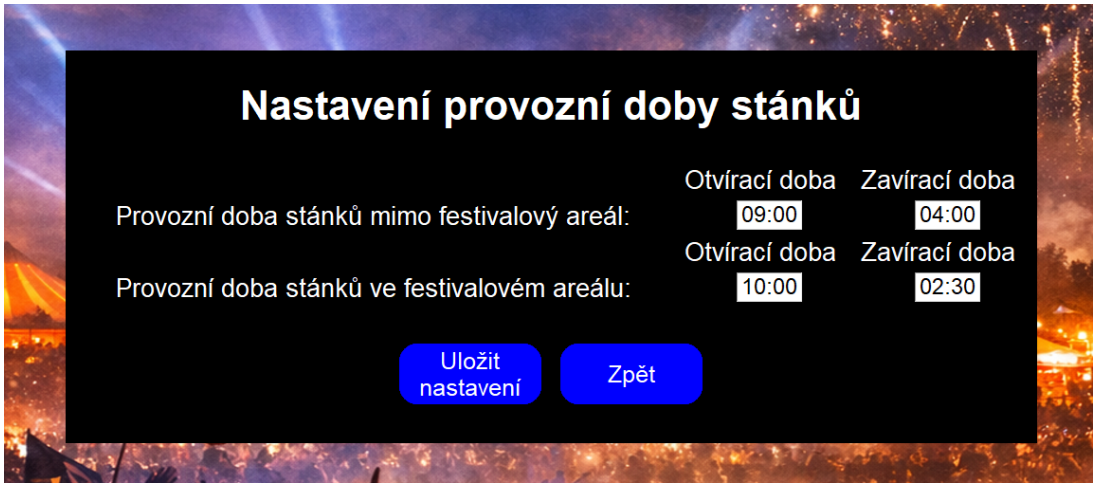
Po otevření nabídky `Kapacity objektů` se zobrazí okno 4, v němž uživatel může nastavit maximální počet osob, které mohou jednotlivé objekty pojmout, případně kolik návštěvníků je daný stánek schopen obsloužit současně. Výjimku tvoří bankomat, u něhož je umožněno provádět výběr vždy pouze jednomu návštěvníkovi.

Ve všech rozšířených nastaveních je nutné upravené hodnoty uložit, aby se skutečně promítly do průběhu simulace. Pokud uživatel změny neuloží, aplikace bude při simulaci pracovat s původními hodnotami. Z tohoto důvodu jsou všechna rozšířená nastavení vybavena dvojicí ovládacích prvků. Prvním z nich je tlačítko `Uložit nastavení`, které provede zápis hodnot do konfiguračního souboru, z něhož jsou data načítána při spuštění simulace. Druhým je tlačítko `Zpět`, jež uživatele vrátí na úvodní obrazovku.



Obrázek 4: Nastavení kapacit objektů

Jednotlivé stánky jsou rozděleny do dvou kategorií, a to na stánky umístěné přímo ve festivalovém areálu a stánky nacházející se mimo areál. Obě skupiny mohou mít odlišnou provozní dobu. Konfigurace provozní doby se provádí v okně zobrazeném na obrázku 5. Uživatel zde zadává čas otevření a uzavření stánků, které poskytují služby návštěvníkům. Výjimkou jsou pokladny, které jsou otevřené vždy hned při startu simulace.



Obrázek 5: Nastavení provozní doby stánků

Cenová konfigurace umožňuje definovat částky, které návštěvníci hradí za

jednotlivé služby poskytované v rámci festivalu. Nastavení cen je dostupné prostřednictvím volby `Ceny festivalu` v základním konfiguračním okně.

Během tvorby festivalového areálu může uživatel umístit do plánku také merch stánek, který návštěvníkům umožňuje zakoupit upomínkové předměty jednotlivých kapel i festivalový merch. V této části nastavení lze konfigurovat cenu každého nabízeného předmětu a zároveň určit počet kusů, které budou během simulace k dispozici. Ve výsledcích simulace budou následně uloženy statistiky prodeje, tedy kolik kusů jednotlivých položek bylo prodáno.

Pro zjednodušení modelu mají všechny kapely jednotnou cenu merche, avšak počet prodávaných předmětů se liší podle popularity kapely. Popularnější kapely si s sebou vezou více předmětů než méně populární kapely.

V rámci nastavení dostupného v okně zobrazeném na obrázku 6 může uživatel konfigurovat barevné označení objektů na festivalovém rozvržení areálu, jehož vytvoření bude popsáno v následujících podkapitolách. Barevné označení je možné u stánků, plochy určené pro stání před pódiem a louky na stanování ve stanovém městečku. Jednotlivé objekty jsou na festivalovém plánku reprezentovány příslušnými emoji ikonami, některé se však nacházejí uprostřed obdélníku. Obdélníkový tvar se používá u objektů, které obsahují další objekty uvnitř, například louka na stanování, kde se v obdélníku nacházejí jednotlivé stany návštěvníků. V takovém případě se zvýraznění provádí vyplněním plochy uvnitř obdélníku zvolenou barvou. Většina ostatních stánků je zobrazena pouze ikonou, u nichž se zvýraznění realizuje vykreslením barevného kruhu pod ikonou. Po najetí kurzoru myši na ikonu objektu se nad objektem zobrazí jeho název.

Zvýraznění stánků je rozděleno do čtyř kategorií: `Stánek je využíván`, `Malá fronta`, `Střední fronta` a `Velká fronta`. První z uvedených kategorií zvýrazní stánek zvolenou barvou ve chvíli, kdy k němu návštěvník přistoupí. Zbývající tři kategorie odpovídají počtu návštěvníků čekajících ve frontě u daného stánku. Uživatel může určit, od jakého počtu osob se má daná úroveň fronty zobrazit, a zároveň zvolit barvu, kterou bude stánek na plánku zvýrazněn. Například lze nastavit, že fronta o velikosti dvaceti osob bude označena jako velká a stánek se v takovém případě zvýrazní červenou barvou. Výběr barvy se provádí pomocí tlačítka `Vybrat`, které otevře dialog pro volbu barvy. Nalevo od tlačítka se nachází náhled aktuálně zvolené barvy. Stejný princip zvýrazňování je použit také u plochy před pódiem a u louky na stanování, kde může uživatel sledovat návštěvnost jednotlivých kapel nebo míru obsazenosti stanových míst. V těchto dvou případech se hodnoty nezadávají jako počet osob, ale jako procentuální obsazenost dané plochy.

Obrázek 6: Nastavení zvýraznění stánků

5.2.2 Nastavení kapel

Po nastavení všech parametrů simulace lze pomocí tlačítka `Pokračovat` přejít k dalšímu kroku, kterým je konfigurace harmonogramu vystoupení kapel. Harmonogram je možné vytvořit ručně, načíst z dříve uloženého souboru, nebo vygenerovat pomocí vestavěného generátoru. Tlačítko `Pokračovat` ve spodním panelu obrazovky je nyní deaktivované a zpřístupní se až v okamžiku vytvoření, vygenerování či načtení časového harmonogramu kapel.

Po kliknutí na tlačítko `Vygenerování lineupu` se uživatel dostane do nabídky konfigurace generátoru časového harmonogramu (obrázek 7). V této části lze nastavit délku vystoupení běžné kapely a *headlinera*, přičemž headlinerem je vždy poslední kapela daného dne. Délka vystoupení headlinera se může v konečném výsledku mírně lišit od hodnoty zadané uživatelem, jelikož přestávky mezi jednotlivými koncerty jsou zaokrouhlovány dolů na nejbližší násobek deseti minut a vzniklý zbytkový čas je následně přičten k délce vystoupení headlinera. Tato úprava zajišťuje přehlednější časový harmonogram, ve kterém koncerty začínají i končí v přirozenější časy.

Uživatel dále může nastavit délku trvání autogramiád, počet kapel generovaných na každý den, čas začátku prvního koncertu a čas ukončení posledního koncertu daného dne. Aplikace umožňuje, aby koncertní blok přesáhl do následujícího dne, přičemž se stále považuje za součást původního festivalového dne. Čas konce posledního koncertu je však omezen nejpozději na 2:00 ráno. Všechna vstupní pole obsahují kontrolu správnosti zadaných hodnot. Při uložení nastavení tlačítkem `Uložit nastavení`, bez něhož se hodnoty do generování nepromítnou, se ověřuje, zda je konfigurace přípustná. Například mezi jednotlivými koncerty musí být zachována minimálně patnáctiminutová přestávka na přestavbu pódia.

Po nastavení všech hodnot uživatel může vygenerovat časový harmonogram pomocí tlačítka `Vygenerovat`. Vygenerování následně umožní uložit harmonogram do zvoleného adresáře tlačítkem `Uložit lineup`.

Obrázek 7: Nastavení generování časového harmonogramu kapel

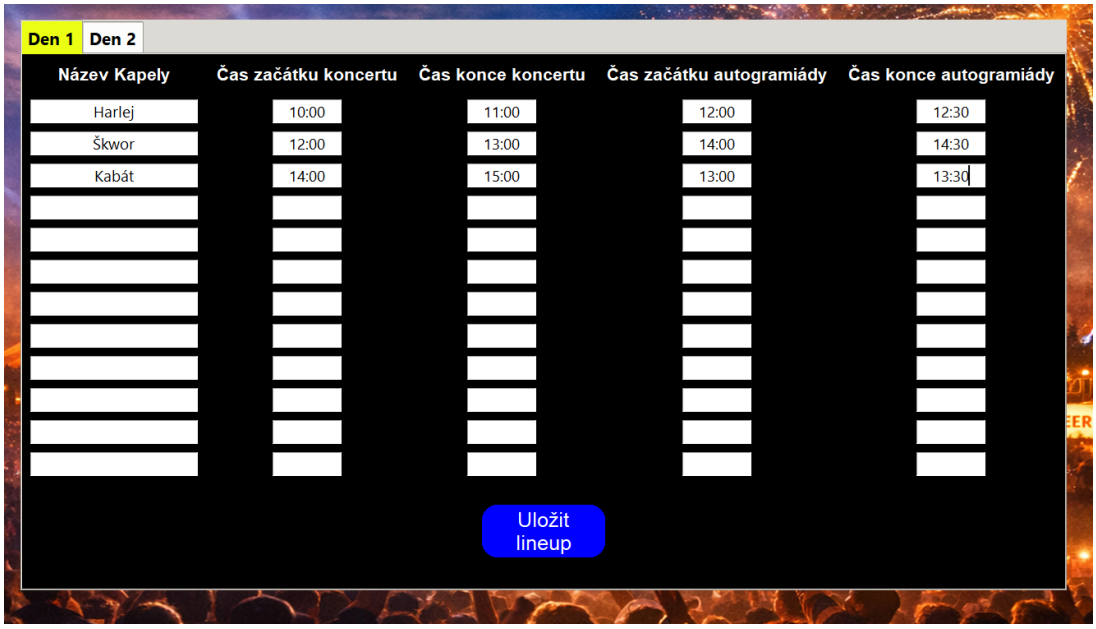
Generátor lineupu využívá seznam kapel uložený v souboru `/data/bands.json`. Každá kapela v tomto souboru obsahuje kromě názvu také atribut `popularity`, který slouží k určení pořadí kapel v rámci jednotlivých festivalových dnů. Při generování se pro každý den náhodně vybere uživatelem zadaný počet kapel, které jsou následně seřazeny podle popularity. Tímto způsobem je určena i role headlinera, jelikož headlinerem se stává kapela s nejvyšší popularitou. Datový soubor je možné libovolně rozšiřovat o další kapely, případně upravit hodnoty popularity.

Vytvoření časového harmonogramu ručně lze v části `Vytvořit lineup`. Tato obrazovka aplikace umožňuje uživateli sestavit časový harmonogram vystoupení kapel manuálně. Rozhraní je rozděleno do jednotlivých festivalových dnů, mezi nimiž lze přepínat pomocí záložek. Každý den obsahuje tabulku, do níž uživatel zadává název kapely, čas začátku a konce koncertu a čas začátku

a konce autogramiády. Tímto způsobem lze vytvořit vlastní lineup bez využití generátoru harmonogramu nebo načítání z externího souboru.

Uživatel může vyplnit libovolný počet řádků, minimálně však jeden a maximálně dvanáct řádků, což odpovídá maximálnímu počtu kapel na den i při generování harmonogramu. Je nutné vyplnit všechny hodnoty na daném řádku, a to i v případě, že uživatel neplánuje na festivalu pořádat autogramiády. Jednotlivé časy se navíc nesmí překrývat. Na obrázku 8 je znázorněno vyplnění formuláře pro první den festivalu, v němž vystoupí pouze tři kapely. Na rozdíl od automatického generování harmonogramu je při ručním vytvoření možné, aby měl každý festivalový den odlišný počet kapel. Při ručním vytváření harmonogramu se neuvádí popularita kapely, protože pořadí kapel i časy jejich vystoupení určuje uživatel. Popularita je však do interního souboru `data/lineup` doplněna při ukládání. První kapela získá hodnotu popularity 10 a každá následující kapela v pořadí má popularitu zvýšenou o 10. Tento mechanismus slouží k tomu, aby kapely s vyšší popularitou měly v simulaci větší množství předmětů ve stánku s merchandisingem.

Jakmile je harmonogram nastavený, je zpřístupněno tlačítko `Pokračovat` v dolním panelu obrazovky. Uživatel tedy může přejít na editaci festivalového areálu.



Obrázek 8: Vytváření časového harmonogramu kapel

5.2.3 Festivalový editor

Festivalový areál je uložen v souboru `data/festival_settings.json`. Pokud se v tomto souboru nachází již dříve vytvořený a validní festivalový areál, je automaticky načten při vstupu do editoru festivalového areálu. V případě,

že soubor neobsahuje žádná data nebo neexistuje, je uživateli zobrazen výchozí prázdný canvas.

Editor se skládá ze dvou levých panelů (viz obrázek 9), centrálního canvasu a spodní lišty s ovládacími tlačítky. Tlačítko `Zpět` uživatele přesměruje na předchozí krok, tedy na nastavení kapel. Pomocí tlačítka `Uložit` lze aktuální podobu areálu uložit. Tlačítko `Načíst` umožňuje načíst již existující areál ze souboru. Funkce `Smazat` vymaže celý canvas a umožní uživateli začít s tvorbou areálu od začátku. Tlačítko `Start` přepne aplikaci do simulačního režimu a zároveň uloží festivalový areál do již zmíněného interního souboru.

Pro úspěšné přepnutí do simulačního režimu je nutné splnit minimální požadavky na strukturu festivalového areálu. Musí existovat zóna `Spawn_zóna`, musí být na festivalu umístěn alespoň jeden stánek prodávající jídlo a nealkoholické nápoje (jakýkoliv stánek s nápoji kromě stánku `Míchané drinky`). Součástí areálu musí být také bankomat, pokladna, toalety (objekt typu `toitoi`) a v zóně `Festivalový areál` musí být pódium. Všechny ostatní objekty jsou volitelné. Poslední podmínkou pro úspěšný přechod do simulačního režimu je propojení jednotlivých zón. Jak splnit tyto požadavky je popsáno v podkapitole 5.2.3.2 `Vytváření rozložení festivalového areálu`. Nejprve si ale představme jednotlivé zóny a jejich účel.

5.2.3.1 Festivalové zóny

Zóny představují logické rozdělení festivalového areálu, přičemž každá z nich má specifický účel. Aplikace poskytuje celkem šest typů zón. `Spawn zóna` je místem, na které návštěvníci při zahájení simulace přicházejí. Tato zóna nemá žádný další účel a není do ní možné umisťovat objekty. `Vstupní zóna` se nachází před hlavním areálem a představuje prostor, do něhož se mohou návštěvníci dostat i bez vstupenky. Slouží jako přirozený předstupeň hlavního areálu a umožňuje oddělit volně přístupné části festivalu od těch, které vyžadují kontrolu vstupních pásků. `Stanové městečko` je určeno pro kempování návštěvníků a poskytuje prostor pro stanování a sprchy. `Zábavní zóna` obsahuje různé atrakce a rozšiřuje nabídku aktivit mimo hlavní hudební program. `Chill zóna` slouží k odpočinku návštěvníků. Může, ale nemusí být umístěna uvnitř festivalového areálu. `Festivalový areál` je hlavní zóna, v níž se nachází pódium, merch stánek, prostor pro autogramiády a další prvky, které vyžadují kontrolu vstupních pásků.

Obrázek 9: Panely pro vytváření festivalového areálu

5.2.3.2 Vytváření rozložení festivalového areálu

Levý sloupec ovládacího panelu obsahuje výběr zón a režimů editace. Pravý sloupec zobrazuje seznam objektů, které je možné do jednotlivých zón umístit. Aktuálně vybraná zóna, režim a objekt jsou zvýrazněny žlutou barvou. Editor nabízí tři režimy práce: `Přidat`, `Editovat` a `Spojit`.

Režim `Přidat` slouží k vytváření zón a vkládání objektů do zón. Každá zóna může být na festivalu pouze jednou a vytváří se kliknutím a tažením myši, čímž vznikne modrý obdélník představující její ohraničení. Vybraný objekt lze umístit pouze do plochy odpovídající zóny a přidává se prostým kliknutím do jejího prostoru.

Režim `Editovat` umožňuje upravovat hranice zóny tažením myši za její hranu a rovněž přesouvat vložené objekty kliknutím, držením a tažením myši. Zóny a objekty vybrané v editačním režimu jsou zvýrazněny červeným ohraničením. V tomto režimu lze také kliknout na spojnici mezi zónami, která se následně rovněž zvýrazní červeně.

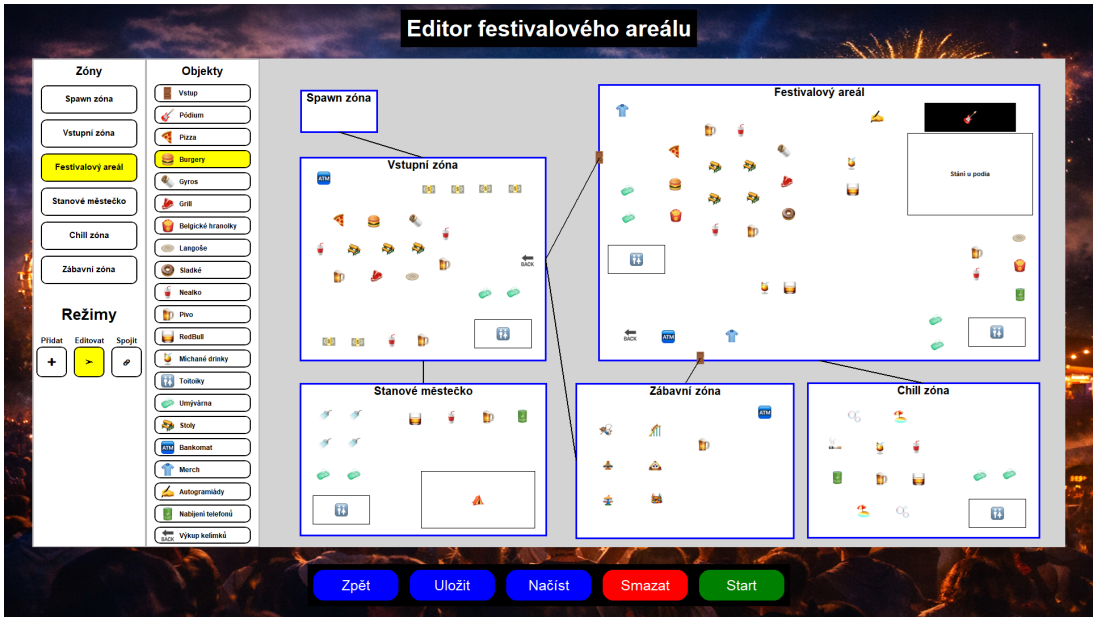
V posledním kroku je nutné vytvořené zóny propojit v režimu `Spojit`. Není nutné spojit každou zónu s každou, ale musí být zajištěno, aby byla každá zóna

dosažitelná, tedy aby vznikl neorientovaný graf. Režim spojení funguje tak, že uživatel nejprve klikne na jednu zónu a poté na druhou. Po druhém kliknutí se mezi nimi vytvoří spojovací čára vedoucí od středu nejbližších hran obou zón.

Zóna `Festivalový areál` může být propojena dvěma způsoby. Prvním je standardní propojení popsané výše, které umožňuje volný pohyb návštěvníků mezi danou zónou a festivalovým areálem. Tento způsob vyjadřuje, že se propojená zóna nachází uvnitř festivalového areálu. Druhým způsobem je propojení přes objekt vstupu. V takovém případě musí návštěvníci při přechodu projít přes vstupní kontrolu, kde se ověřuje, zda mají vstupní pásek. Pro tento typ propojení je nutné v režimu `Přidat` vložit do festivalového areálu objekt `Vstup`, který lze umístit pouze na hranu zóny `Festivalový areál`. Následně lze v režimu `Spojit` kliknout na vstup a poté na zónu, kterou má vstup propojit.

Objekt vstupu lze v editačním režimu přesouvat po hraně zóny pouze tehdy, pokud není spojen s žádnou jinou zónou. Hranou zóny lze pohybovat pouze v případě, že na ní není umístěn žádný vstup a zároveň se v její těsné blízkosti nenachází žádný objekt.

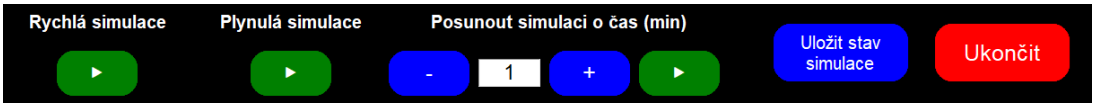
Vybraný objekt, zónu nebo spojení mezi zónami lze smazat stiskem klávesy `Delete`. Smazáním vstupu se odstraní všechna jeho propojení. Smazáním zóny se odstraní všechny objekty uvnitř zóny i všechna její spojení. Na obrázku 10 je ukázka možného výsledného rozložení festivalového areálu. Po vytvoření areálu můžeme pokračovat do simulační části aplikace stiskem tlačítka `Start`.



Obrázek 10: Ukázka možného rozložení festivalového areálu

5.2.4 Simulační režim

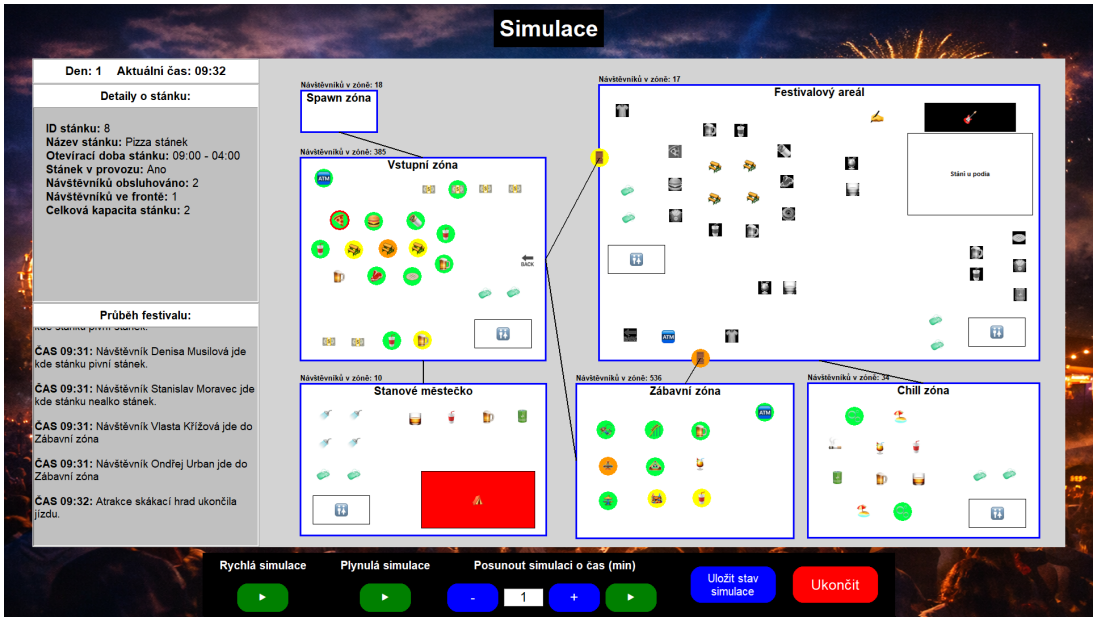
Při přechodu do simulačního režimu probíhá generování návštěvníků a vytváření entit potřebných pro běh simulace. Z tohoto důvodu může při větším počtu návštěvníků přepnutí trvat několik vteřin. V simulačním režimu se u každé zóny v levém horním rohu zobrazí ukazatel aktuálního počtu návštěvníků v dané zóně. Změní se také levý panel. V jeho záhlaví najdeme informaci o aktuálním simulačním čase a aktuálním festivalovém dnu. Pod záhlavím se nachází horní část panelu, ta nyní zobrazuje detailní informace o jednotlivých objektech. Spodní část slouží jako výpis událostí, které se v simulaci právě odehrávají. Detail konkrétního objektu si uživatel zobrazí kliknutím na jeho ikonu. V simulačním režimu není možné měnit rozmístění objektů ani jinak upravovat rozložení areálu. Současně se změní i spodní ovládací lišta, jejíž simulační verzi lze vidět na obrázku 11. Obsahuje mimo jiné tlačítko `Ukončit`, které simulaci ukončí a vrátí uživatele zpět do editoru.



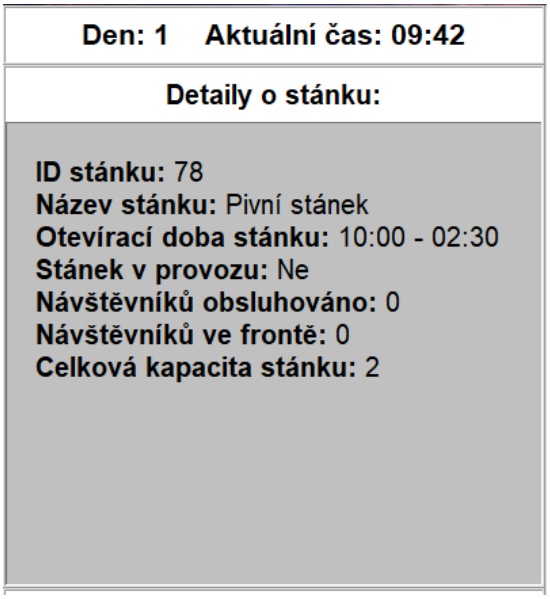
Obrázek 11: Spodní lišta v simulačním režimu

Spodní lišta nabízí dva základní způsoby spuštění simulace. Prvním z nich je rychlá simulace aktivovaná tlačítkem `Rychlá simulace`. V tomto režimu proběhne celá simulace zrychleně na pozadí a uživatel si následně může zobrazit výsledky simulace dostupné ve složce `outputs`.

Druhým způsobem je plynulá simulace, která se spouští tlačítkem `Plynulá simulace`. V tomto režimu běží simulace v reálném čase, kde jedna minuta simulace odpovídá jedné vteřině skutečného času. Uživatel může průběžně sledovat měnící se počty návštěvníků v jednotlivých zónách, barevné zvýraznění objektů při vzniku front či jejich vytíženosti, detaily objektů a také živé výpisy událostí ze simulace. Výpisy jsou zobrazovány se zpožděním, jelikož vznikají během simulace dané minuty, ale jejich vykreslení probíhá až při následující aktualizaci zobrazených dat. Na obrázku 12 je zachycen začátek plynulé simulace při nastavení festivalu na dva dny, začátek simulace v 8:00 a s celkovým počtem jednoho tisíce návštěvníků. V levém panelu je v horní části zobrazen detail `Pizza stánku`, u něhož se aktuálně nachází jeden návštěvník ve frontě a dva návštěvníci jsou právě obsluhováni. Zajímavým detailem je červené zvýraznění louky na stanování ve stanovém městečku, které signalizuje vyčerpání kapacity pro umístění dalších stanů. V zóně `festivalový areál` lze zároveň pozorovat, že stánky obsluhující návštěvníky mají šedé ikony, což značí, že jsou v aktuálním čase simulace zavřené. Po otevření detailu stánku `Pivní stánek` nacházejícího se v zóně `Festivalový areál` můžeme vidět, že otevření tohoto stánku nastane za osmnáct minut simulovaného času (viz obrázek 13).



Obrázek 12: Ukázka plynulé simulace



Obrázek 13: Ukázka plynulé simulace

Lišta dále nabízí možnost posunout simulaci o zadaný čas prostřednictvím rozhraní `Posunout simulaci o čas`. To obsahuje vstupní pole přijímající pouze celočíselné hodnoty. Po stisknutí zeleného tlačítka se simulace posune o zadaný počet minut dopředu. Hodnotu je možné upravovat také pomocí tlačítek po stranách vstupního pole, která umožňují její zvýšení či snížení o jednu jednotku.

V průběhu plynulé simulace není možné přejít na rychlou simulaci a posouvat simulační čas prostřednictvím rozhraní `Posunout simulaci o čas`, s výjimkou tlačítka `Plynulá simulace`, kterým lze simulaci pozastavit. Uživatel může simulaci kdykoliv spustit, zastavit, prozkoumat aktuální stav a opět pokračovat. Zbytek simulace lze případně dokončit zrychleně pomocí rychlé simulace, pokud je simulace pozastavená.

Během plynulé simulace lze kdykoliv uložit aktuální stav simulace do souboru `outputs/simulation_states.json` pomocí tlačítka `Uložit stav simulace`.

Po dokončení simulace se zobrazí informativní zpráva oznamující její úspěšné dokončení. Po stisknutí tlačítka `Zavřít` je uživatel přesměrován zpět na úvodní obrazovku, odkud může zahájit novou simulaci.

6 Výsledky simulace

Výsledky simulace jsou po jejím dokončení zpracovány funkcí `end_of_simulation()`. Tato funkce informuje uživatele o úspěšném ukončení simulace a následně předá veškerá nasbíraná data z kontroleru modulu `/outputs/code/logs.py`, který zajišťuje jejich zpracování a uložení. Modul provede agregaci průběžně zaznamenaných statistik a uloží všechny výstupy do samostatných `json` souborů v adresáři `/outputs`. Tyto soubory jsou při každém novém spuštění simulace přepsány aktuálními výsledky.

Soubor `all_simulation_logs.json` obsahuje kompletní chronologický výpis všech událostí, které během simulace nastaly, tedy jak individuální události návštěvníků, tak i globální události festivalu, například začátek koncertu nebo otevření stánku.

Soubor `groups_and_members_logs.json` ukládá detailní informace o skupinách návštěvníků a o jednotlivých členech těchto skupin. U každé skupiny je uveden její typ, počet členů, počet dětí, společná historie aktivit a následně výpis všech členů. Každý člen má uvedené osobní údaje, tedy jméno, příjmení, pohlaví, věkovou kategorii a věk, dále své vlastnosti, preference, seznam kapel, od kterých získal autogram, zakoupený merch, kompletní výpis svých událostí, který se může lišit od skupinového, pokud se návštěvník od skupiny dočasně odpojil a také přehled všech výdajů. Na závěr má každý návštěvník uložené své osobní výpisy událostí, které umožňují sledovat jeho individuální průchod festivalem.

Soubor `merch_stats.json` obsahuje statistiky prodeje upomínkových předmětů. U každého typu předmětu je uveden počet prodaných kusů a celkový zisk z prodeje.

Soubor `simulation_states.json` ukládá stavy simulace uložené uživatelem během plynulé simulace tlačítkem `uložit stav simulace`. Každý uložený stav obsahuje aktuální čas simulace a přehled počtu návštěvníků v jednotlivých zónách, spolu s aktuálním stavem každého stánku.

Soubor `stalls_stats.json` obsahuje statistiky všech stánků na festivalu.

Každý obslužný stánek obsahuje informaci o průměrné a maximální délce fronty, největším počtem čekajících návštěvníků, průměrné čekací době a o celkovém počtu obsloužených osob.

Soubor `standing_by_stage_stats.json` obsahuje informace a celkovém a průměrném počtu návštěvníků na koncertě každé kapely.

Nakonec soubor `zones_stats.json` zachycuje vytížení jednotlivých zón v průběhu simulace, tedy průměrný počet návštěvníků v zóně, maximální dosaženou obsazenost a konkrétní časy, kdy byla zóna nejvíce zaplněná.

7 Možná rozšíření

Aplikace v současné verzi využívá několik zjednodušení, která byla zvolena s ohledem na rozsah práce a přehlednost výsledného modelu. Je však zřejmé, že by bylo možné doplnit mnoho dalších funkcí a objektů, které by model dále rozšířily. Uvedený seznam možných rozšíření proto není konečný a představuje pouze nejvýznamnější směry, které se během vývoje nabízely.

- **Více typů zón** – Aktuální verze aplikace umožňuje použít každý typ zóny pouze jednou. Do budoucna by bylo možné rozšířit editor o více instancí stejných zón, například několik samostatných chill zón, více vstupních zón nebo více stanových městeček. Zároveň by bylo možné přidat i nové typy zón, jako například VIP zónu, technické zázemí, backstage, parkoviště nebo zónu pro zdravotní službu. Tím by bylo možné modelovat rozsáhlejší a realističtější festivalové areály.

- **Více typů objektů** – Současná implementace obsahuje základní sadu objektů nezbytných pro běh simulace a typických pro téměř jakýkoliv hudební festival. Model by bylo možné rozšířit o další objekty, které by nabízely návštěvníkům více dostupných aktivit. Zároveň je v aktuální verzi možné umístit pouze jedno pódium. Rozšíření o více pódií by umožnilo simulovat paralelní koncertní program.

- **Prvky infrastruktury** – V aktuální verzi simulace nejsou zahrnuty prvky interní infrastruktury festivalu, které mají zásadní vliv na jeho provoz. Do budoucna by bylo možné doplnit například zaměstnance festivalu, jako jsou členové bezpečnostní služby, zdravotníci, technici, dobrovolníci či pracovníci úklidu. Další možností je přidání objektů typu backstage, technické zázemí, skladovací prostory nebo servisní stanoviště. Tyto prvky by umožnily simulovat logistiku festivalu, pohyb personálu a jejich interakci s návštěvníky.

- **Ekonomický model simulace** – Současná verze aplikace pracuje pouze se základními cenami služeb a merchandisingu. Rozšíření o komplexnější ekonomický model by umožnilo sledovat finanční stránku festivalu, například celkové náklady na kapely, provoz stánků, pronájem areálu či mzdy zaměstnanců. Zároveň by bylo možné vyhodnocovat zisk jednotlivých stánků,

celkový výnos festivalu, vliv nastavení cen na chování návštěvníků. Takové rozšíření by poskytlo detailnější pohled na ekonomickou udržitelnost festivalu.

- **Vnější vlivy** – Simulaci by bylo možné rozšířit o faktory, které ovlivňují průběh festivalu z vnějšího prostředí. Patří sem například počasí, které může ovlivnit návštěvnost, délku front nebo využití jednotlivých zón. Dalším příkladem jsou mimořádné události, bezpečnostní incidenty či technické poruchy. Tyto vlivy by umožnily simulovat krizové scénáře a testovat odolnost festivalového provozu.

- **Optimalizace vizuálního běhu simulace** – Plynulá simulace v aktuální implementaci průběžně aktualizuje vizuální stav celého festivalu, mění barvy a ikony objektů podle jejich vytížení, zobrazuje výpisy událostí a průběžně aktualizuje počty lidí v jednotlivých zónách. V kombinaci s Tkinterem, který není navržen pro časté a rozsáhlé aktualizace velkého množství grafických prvků, vede tento přístup k výraznému zatížení hlavního vlákna aplikace. Výsledkem je snížená plynulost, opožděné vykreslování a celkové sekání simulace při vyšším počtu návštěvníků. Oproti tomu rychlá simulace probíhá bez grafického rozhraní a díky absenci vizuálního vykreslování běží i při vysokém počtu návštěvníků zcela plynule. Možným rozšířením by mohlo být využití modernějších grafických knihoven.

Závěr

V této práci jsem navrhl a implementoval simulační model hudebního festivalu založený na principu diskrétní simulace. Diskrétní simulace je vhodná zejména pro prostředí, kde se jednotlivé akce odehrávají v konkrétních časových okamžicích a mají přímý vliv na stav celého systému. Principy diskrétní simulace jsem využil při vytváření simulačního modelu hudebního festivalu pomocí knihovny SimPy. Knihovna umožňuje přesně definovat události, jejich časové trvání, vzájemné závislosti a reakce systému na změny v prostředí. Vytvořený model tak dokáže realisticky zachytit chování návštěvníků, jejich přesuny mezi zónami, vznik a rozpouštění front u stánků i dynamiku skupinových aktivit. Současně umožňuje sledovat vytížení jednotlivých částí festivalu, efektivitu obsluhy stánků, časové špičky u pódia a celkový tok návštěvníků v průběhu dne. Díky podrobnému logování a ukládání výsledků do strukturovaných souborů lze následně provádět detailní analýzu průběhu simulace. Přestože některé části simulace mohou být dále rozšířeny nebo zpřesněny, vytvořený systém ukazuje, že diskrétní simulace je vhodným nástrojem pro modelování komplexního prostředí hudebního festivalu a může sloužit jako základ pro další vývoj, experimentování i praktické využití při plánování podobných akcí.

Conclusions

In this thesis, I designed and implemented a simulation model of a music festival based on the principles of discrete-event simulation. Discrete-event simulation is particularly suitable for environments in which individual actions occur at specific points in time and have a direct impact on the state of the entire system. I applied these principles when creating a simulation model of a music festival using the SimPy library. The library makes it possible to precisely define events, their duration, mutual dependencies, and the system's responses to changes in the environment.The resulting model is capable of realistically capturing visitor behaviour, their movement between zones, the formation and dissolution of queues at stalls, and the dynamics of group activities. At the same time, it enables monitoring of the load on individual parts of the festival area, the efficiency of stall service, peak times at the stage, and the overall flow of visitors throughout the day. Thanks to detailed logging and storing results in structured files, it is possible to perform an in-depth analysis of the simulation's course. Although some parts of the simulation could be further extended or refined, the developed system demonstrates that discrete-event simulation is an appropriate and effective tool for modelling the complex environment of a music festival and can serve as a foundation for further development, experimentation, and practical use in planning similar events.

A Obsah elektronických dat

Elektronická data odevzdaná v systému KI PřF UP v Olomouci obsahují veš keré soubory k textu práce, včetně zdrojových kódů potřebných ke zprovoznění aplikace. Struktura elektronické přílohy je následující:

`text/`
: Adresář s textem práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech (textových) příloh, a všechny soubory potřebné pro bezproblémové vytvoření PDF dokumentu textu (případně v ZIP archivu), tj. zdrojový text textu a příloh a vložené obrázky.

`README.txt`
: Textový soubor obsahující postup ke spuštění aplikace.

`festival_simulator/`
: Adresář se zdrojovými kódy a daty potřebnými k běhu aplikace. V podadresáři `/outpusts` se nachází výstupy ze simulace.

`assets/`
: Adresář obsahuje `json` soubory představující předpřipravené konfigurace festivalového areálu a připravené časové harmonogramy vystoupení kapel, které lze v aplikaci načíst bez nutnosti jejich vytváření.

`references/`
: V tomto adresáři se nachází studijní materiál poskytnutý doktorem Laštovičkou, který sloužil jako jeden ze zdrojů při tvorbě této práce.

Literatura

[1] Banks, Jerry; Carson, John S.; Nelson, Barry L.; Nicol, David M. *Discrete-Event System Simulation*. 5. vyd. Upper Saddle River: Pearson, 2013. Dostupný také z: ⟨https://ferltz.github.io/inv_oper_2/documents/books/Discrete-Event%20System%20Simulation-Pearson%20Banks%20Carson%20(2013).pdf⟩. ISBN 978-1-292-02437-0.

[2] *Emoji Generator*. Accessed: 25 July 2026. Dostupný z: ⟨https://emoji.aranja.com⟩.

[3] Developers, SimPy. *SimPy Documentation*. 2024. [online; cit. 2026-07-20]. Dostupný z: ⟨https://simpy.readthedocs.io/⟩.

[4] Mgr. Jan Laštovička, Ph.D. *KMI/UPS - lectures10, Iterátory*. 2024. studijní materiál.

[5] Foundation, Python Software. *Tkinter — Python Standard GUI Library*. 2026. [online; cit. 2026-07-20]. Dostupný z: ⟨https://docs.python.org/3/library/tk.html⟩.

[6] Schimansky, Tom. *CustomTkinter Documentation*. 2025. [online; cit. 2026-07-20]. Dostupný z: ⟨https://customtkinter.tomschimansky.com/documentation/⟩.

[7] Clark, Alex; Contributors. *Pillow Documentation*. 2026. [online; cit. 2026-07-20]. Dostupný z: ⟨https://pillow.readthedocs.io/en/stable/⟩.