Príloha A: Obsah digitálnej časti práce

Digitálna časť práce obsahuje zdrojový kód vo forme Python projektu v pôvodnej forme a taktiež v inštalovateľnom balíčku typu wheel, pre inštaláciu projektu ako knižnica.

Obsiahnuté sú tu tiež všetky zdroje programu, t.j. grafické prvky, súčasti miestností a písma. Tieto súbory sa nachádzajú v priečinku `src/pg_gen/assets`. Všetky použité zdroje sú pod licenciou, ktorá umožňuje ich ďalšiu distribúciu. Kópia alebo odkaz na relevantnú licenciu je umiestnený popri súbore, na ktorý sa vzťahuje.

- Zdrojový kód sa nachádza v priečinku: `PlatformGameGenerator`
- Inštalovateľný balíček je súbor: `pg_gen-1.0.0-py3-none-any.whl`

Zdrojový kód obsahuje súbor `pyproject.toml` vyjadrujúci knižnice a verziu Pythonu, potrebnú pre jeho spustenie. Na manažment knižníc bol použitý program `rye` verzie `0.44.0`, ktorý je možné stiahnuť cez internet z adresy `https://rye.astral.sh/`. Po inštalácií danej aplikácie je možné nainštalovať potrebné knižnice príkazom `rye sync`. Tento príkaz vytvorí Python virtuálne prostredie, ktoré je možné aktivovať príkazom vhodným pre Váš operačný systém.

Projekt definuje príkazy, ktoré služia na spustenie komponentov práce bez potreby písania ďalšieho kódu. Tieto príkazy je možné spustiť po aktivácií Python prostredia.

- `start` ⇒ vygeneruje úroveň a sputí interaktívnu hru
- `start-editor <súbor>` ⇒ spustí editor a otvorí súčasť miestnosti definovanú v špecifikovanom súbore

Príloha B: Návod na použitie programu

Inštalácia programu ako knižnica

Aby bol program použiteľný ako knižnica, je potrebné ho nainštalovať z wheel súboru. Tento súbor je súčasťou C3 alebo je k dispozícií na internetovej adrese `https://bt7s7k7-2.github.io/PlatformGameGenerator/pg_gen-1.0.0-py3-none-any.whl`. Tento súbor, kedže je štandardným formátom pre Python knižnice, je možné nainštalovať prostredníctvom Vami zvoleného inštalačného programu. Nižšie sú uvedené príklady inštalácie prostredníctvom niektorých programov.

```
pip install 'pg_gen-1.0.0-py3-none-any.whl'
rye add pg_gen --path 'pg_gen-1.0.0-py3-none-any.whl'
```

Použitie programu priamo

Program potrebuje pre svoju činnosť zdroje. Tieto sú manažované globálnymi objektami, ktoré treba inicializovať.

```
import pygame
from pg_gen.generation.RoomPrefabRegistry import RoomPrefabRegistry
from pg_gen.level_editor.ActorRegistry import ActorRegistry

pygame.init()
ActorRegistry.load_actors()
RoomPrefabRegistry.load()
```

Ďalším krokom je zvoliť si obtiažnosť, ktorú požadujete.

```
from pg_gen.generation.RoomParameter import UNUSED_PARAMETER
from pg_gen.difficulty.DifficultyReport import DifficultyReport

target_difficulty = DifficultyReport()
target_difficulty.set_all_parameters(UNUSED_PARAMETER)
target_difficulty.set_parameter(RoomParameter.REWARD, 500)
target_difficulty.set_parameter(RoomParameter.JUMP, 10)
target_difficulty.set_parameter(RoomParameter.ENEMY, 100)
target_difficulty.set_parameter(RoomParameter.SPRAWL, 50)
```

Následne je potrebné vytvoriť optimalizátor a vložiť zvolené parametre. Optimalizátor ďalej príjima argumenty `random`, kde je potrebné vložiť náhodný generátor - tu je možné nastaviť konzistentný seed. Optimalizátor má možnosť nastaviť populáciu jedincov genetického algoritmu cez parameter `max_population` a počet iterácií genetického algoritmu cez parameter `max_generations`.

```
from random import Random
from pg_gen.game_core.Universe import Universe
from pg_gen.difficulty.DifficultyOptimizer import DifficultyOptimizer

universe = Universe()
optimizer = DifficultyOptimizer(universe, target_difficulty=target_difficulty, random=Random(108561))
```

Nie je potrebné špecifikovať všetky parametre. Nepoužité parametre ostanú ako `UNUSED_PARAMETER`. Tieto parametre budú mať po optimizácií náhodnú hodnotu, preto je vhodné nastaviť ich nastaviť na konštatnú hodnotu. Takto je možné nastaviť všetky parametre, ktoré sa menia pri optimalzácií.

```
from pg_gen.generation.RoomParameter import RoomParameter

optimizer.get_parameter(RoomParameter.REWARD).override_value(0.5)
```

Po zvolení parametrov a obtiažnosti je možné spustiť optimalizátor.

```
optimizer.initialize_population()
optimizer.optimize()
```

Po optimizácií je možné vybrať najvhodnejšieho kandidáta a použiť jeho úroveň. Následne je potrebné aktivovať koreňovú miestnosť a vložiť do nej objekt hráča.

```
from pg_gen.generation.RoomController import RoomController
from pg_gen.actors.Player import Player
from pg_gen.support.Point import Point
from pg_gen.support.constants import ROOM_HEIGHT, ROOM_WIDTH

best_candidate = optimizer.get_best_candidate()
map = best_candidate.get_map()
universe.map = map

room_controller = RoomController.initialize_and_activate(universe, map.get_room(Point.ZERO), None)
room_controller.world.add_actor(Player(position=Point(ROOM_WIDTH / 2, ROOM_HEIGHT / 2)))
```

Pre použitie hry interaktívne, stači spustiť `InteractiveGameLoop`.

```
from pg_gen.game_core.InteractiveGameLoop import InteractiveGameLoop

game_loop = InteractiveGameLoop(universe)
game_loop.run()
```

Pre použitie s modelom, je potrebné vytvoriť vlastný pygame-ový `Surface` a vložiť ho do `GameLoop`. Následne je možné vkladať vstup cez objekt `InputState` a manuálne simulovať update. Keďže mimo interaktívneho prístupu nie je herná slučka naviazaná na reálny čas, je potrebné stanoviť čas, ktorý prejde medzi každý update krokom. Napríklad: ak chceme mať 20 krokov cez jednu simulovanú sekundu, tak čas medzi krokmi je $1/20$ sekúnd.

```
from pg_gen.game_core.GameLoop import GameLoop
from pg_gen.support.constants import CAMERA_SCALE
from pg_gen.game_core.InputState import InputState

surface = pygame.display.set_mode((CAMERA_SCALE * ROOM_WIDTH, CAMERA_SCALE * ROOM_HEIGHT))
self.game_loop = GameLoop(surface, self.universe)

input_state = self.universe.di.inject(InputState)
input_state.clear()

input_state.left = True
input_state.jump = True
```

```
# [...]

game_loop.update_and_render(1 / fps)
```

Použitie Gymnasium prostredia

Gymnasium prostredie rieši spustenie hernej slučky automaticky. Stačí inicializovať globálne objekty a vložiť úroveň. Prostredie je automaticky registrované, stačí importovať package `pg_gen_gym`.

```
import pg_gen_gym
```

Následne je možné vytvoriť prostredie.

```
env = gymnasium.make("gymnasium_int/PgEnv", render_mode="rgb_array", level=level)
observation, info = env.reset()
```

Prostredie podporuje dva render módy:

- `human` ⇒ simulácia prebieha v reálnom čase a je otvorené okno, kde je možné vidieť výstup z hry
- `rgb_array` ⇒ simulácia prebieha najrýchlešie ako je možné, výstupom modelu je array obsahujúci grafický výstup hry

Pri použití `rgb_array`, kedže sa neotovrí okno, je potrebné manuálne inicializovať grafický systém pygame-u. Najjednoduchší spôsob, ako to dosiahnuť, je vytvoriť jednopixelové skryté okno, ktoré nebude na nič využité.

```
if env.render_mode == "rgb_array":
    pygame.display.set_mode((1, 1), flags=pygame.HIDDEN)
```

Ako parameter `level` je možné dodať:

- string ⇒ ako explicitný názov miestnosti, ktorá bude vygenerovaná; toto je možné použiť pre testovanie konkrétnej mechaniky
- callback ⇒ funckia dostane ako argument referenciu na `Universe` a musí vrátiť `Map` objekt; vo funkcií spustite generáciu ako v predošlej sekcii

Toto je jednoduchý loop, pre využitie prostredia:

```
episode_over = False
while not episode_over:
    action = env.action_space.sample()
    observation, reward, terminated, truncated, info = env.step(action)

    episode_over = terminated or truncated
    print(action, observation, info, reward)

env.close()
```

Príloha C: Harmonogram práce

Zimný semester

Cieľom práce v zimnom semestri bolo vytvoriť základ generovania úrovní. Preto bolo potrebné vytvorenie komponentu Generátor úrovne, na jeho podporu komponent Editor súčastí a na jeho vyskúšanie komponent Hra.

Týždeň	**Plán aktivít**
1.	Plánovanie hernej architektúry
2.	Práca na hernom komponente do stavu možnosti prechodu cez miestnosti hernou postavou
3.	Práca na generovaní úrovne do stavu generovania rozloženia miestností
4.	Práca na editore do stavu možnosti tvorby koreňových súčastí
5.	Implementácia herných mechaník
6.	Implementácia herných mechaník
7.	Implementácia herných mechaník
8.	Implementácia herných mechaník
9.	Práca na generovaní miestností a kombinovanie súčastí
10.	Práca na generovaní miestností a kombinovanie súčastí
11.	Práca na generovaní miestností a kombinovanie súčastí
12.	Práca na generovaní miestností a kombinovanie súčastí
13.	Práca na odovzdaní BP1

V rámci práce na implementácií herných mechaník a generovaní miestností, je aj práca na funkcionalite editora. Nové funkcie boli pridané práve vtedy, keď boli potrebné pre testovanie súčasti, na ktorej sa práve pracovalo. Práca na odovzdaní BP1 pokračovala aj po skončení výučby počas prázdnin.

Letný semester

Cieľom letného semestra je práca na optimalizátore obtiažnosti. Súčasťou je aj implementácia podporných systémov optimalizátora, t.j. meranie obtiažnosti miestností a nájdenie optimálnej cesty cez úroveň. Tiež je dôležitou súčasťou tiež práca na záverečnej správe projektu.

Týždeň	Plán aktivít
1.	Plánovanie architektúry optimalizatora
2.	Plánovanie architektúry optimalizatora
3.	Implementácia podporných systémov optimalizátora
4.	Implementácia podporných systémov optimalizátora
5.	Implementácia podporných systémov optimalizátora
6.	Implementácia podporných systémov optimalizátora
7.	Práca na optimalizátore obtiažnosti
8.	Práca na optimalizátore obtiažnosti
9.	Práca na optimalizátore obtiažnosti
10.	Práca na optimalizátore obtiažnosti, Práca na záverečnej správe projektu
11.	Implementácia Gymnasium prostredia, Práca na záverečnej správe projektu
12.	Práca na záverečnej správe projektu

Kontrola a overovanie funkčnosti projektu prebiehala zároveň s vývojom funckionality. Finálne overenie funkcionality prebehlo popri písaní záverečnej správy projektu.