PAVOL JOZEF ŠAFÁRIK UNIVERSITY IN KOŠICE
FACULTY OF SCIENCE

WARLOCK (FOR NOW)

Master's thesis

2023 **Bc. Dominik Džama**

PAVOL JOZEF ŠAFÁRIK UNIVERSITY IN KOŠICE
FACULTY OF SCIENCE

WARLOCK (FOR NOW)

Master's thesis

Study Programme:	Informatics
Study Field:	18. Informatics
Institute:	Institute of Computer Science
Supervisor:	RNDr. Juraj Šebej, PhD.

Košice 2023 **Bc. Dominik Džama**

Contents

1 Názvoslovie

1.1 Unity

Unity alebo Unity Engine je herný engine vyvinutý spoločnosťou Unity Technologies. Je to výkonný a veľmi populárny engine používaný na vývoj hier, simulácií, virtuálnej reality a ďalších interaktívnych aplikácií. Unity podporuje skriptovanie hier a aplikácií pomocou rôznych jazykov, vrátane C#. Skripty a ďalšie komponenty sa pripájajú k herným objektom a slúžia na definovanie ich správania, logiky a interakcie.[2].

1.1.1 Game object

- je najdôležitejším konceptom v Unity Editori.
- Každý objekt v hre je GameObject, od postáv a zbierateľných predmetov po svetlá, kamery a špeciálne efekty.

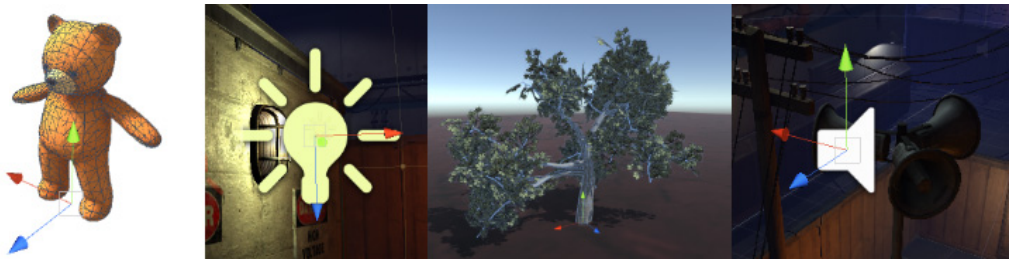

Fig. 1: Rôzne príklady game objektov

- GameObjecty sú základné objekty v Unity, ktoré predstavujú postavy, objekty a scenériu. Samy osebe nevykonávajú veľa, ale slúžia ako kontajnery pre komponenty, ktoré implementujú funkcionalitu.
- Unity má veľa rôznych vstavaných typov komponentov a tiež je možné vytvoriť vlastné komponenty pomocou Unity Scripting API.
- Každý GameObject vždy má pripojený komponent Transform (pre reprezentáciu pozície, orientácie a mierky) a nie je možné ho odstrániť. Ostatné komponenty,

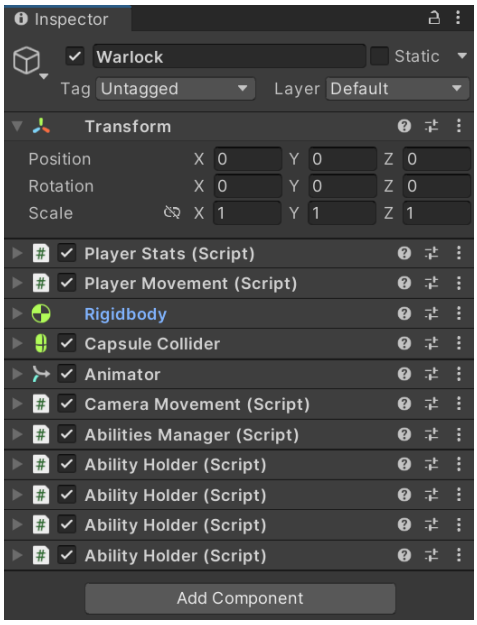

Fig. 2: Viacero typov komponentov na GameObjecte "Warlock"

ktoré dávajú objektu jeho funkcionalitu, je možné pridať [6].

1.1.2 Transform

V rámci Unity je každý herný objekt reprezentovaný komponentom Transform, ktorý je pripojený k danému objektu. Transform obsahuje nasledujúce vlastnosti:

- Position (pozícia) Určuje 3D pozíciu objektu v priestore. Pozícia je reprezentovaná ako vektor s hodnotami X, Y a Z.
- Rotation (rotácia) Určuje rotáciu objektu vo forme Eulerových uhlov alebo ako kvaternión. Rotácia je definovaná ako otáčanie objektu okolo osí X, Y a Z.
- Scale (mierka) Určuje veľkosť objektu na jednotlivých osiach. Mierka je tiež reprezentovaná ako vektor s hodnotami X, Y a Z. Zmena mierky ovplyvňuje veľkosť objektu a všetkých jeho podobjektov.

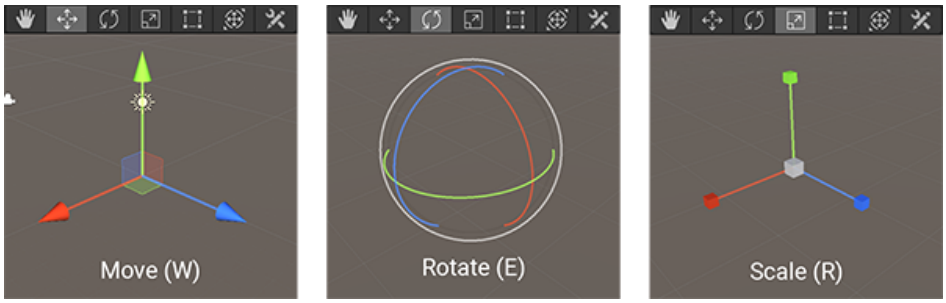

Fig. 3: Transform vlastnosti vizualizované v editori (position, rotation, scale)

- Hierarchia Umožňujúci nastavenie rodičovského objektu pre jednotlivé objekty a vytváranie hierarchického usporiadania medzi nimi.

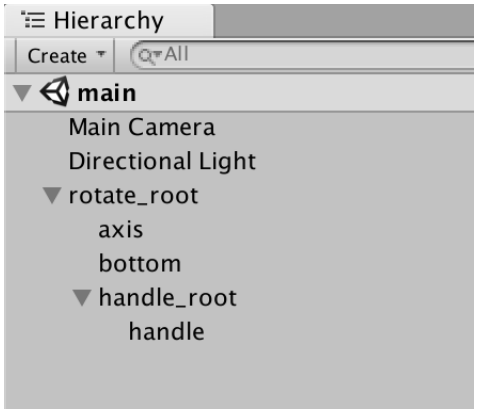

Fig. 4: Transform - uchovávateľ hierarchie

Transform komponenta tiež poskytuje rôzne metódy a funkcie na manipuláciu s objektom:

- Translate: Posunie objekt o daný vektor v priestore.
- Rotate: Otočí objekt o daný uhol okolo určenej osi.
- LookAt: Nastaví rotáciu objektu tak, aby sa smeroval k zadanému cieľu.
- SetParent: Nastaví rodičovský objekt, čo umožňuje hierarchické usporiadanie objektov.

Transform komponent je dôležitou súčasťou herného objektu v Unity, pretože umožňuje transformovať a manipulovať jeho pozíciou, rotáciou a mierku. To je kľúčové pre pohyb, animácie, kolízie a iné aspekty herného systému.

1.1.3 Collider

V Unity je Collider komponent, ktorý slúži na detekciu kolízií medzi objektmi v hernom svete. Collider definuje tvar a rozsah kolízie pre daný objekt a umožňuje detekciu kontaktu a interakciu s inými objektmi.

Collider je pripojený k hernému objektu a definuje oblasť, v ktorej sa vykonáva detekcia kolízie. Existuje niekoľko základných typov:

- Box Collider: Reprezentuje kolíznu oblasť vo forme kvádra. Je vhodný pre objekty s pravidelným tvarom, ako sú steny alebo krabice.

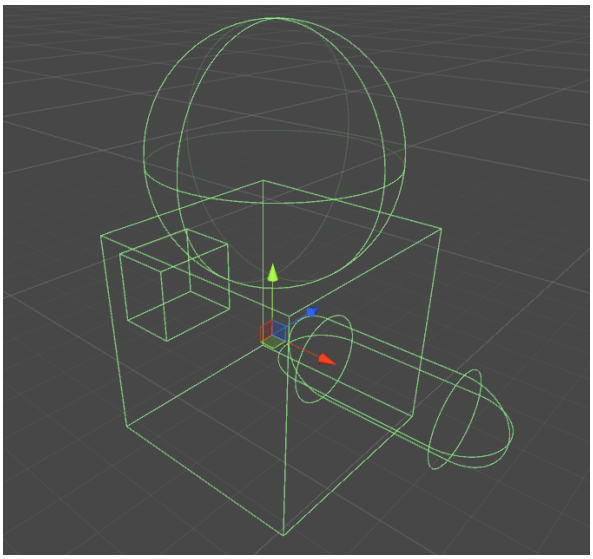

Fig. 5: Rôzne typy Colliderov.

- Sphere Collider: Reprezentuje kolíznu oblasť vo forme gule. Je vhodný pre objekty s guľovým tvarom, ako sú lopty alebo planéty.
- Capsule Collider: Reprezentuje kolíznu oblasť vo forme valca s polokruhovými kónmi na koncoch. Je vhodný pre objekty s valcovým tvarom, ako sú postavy alebo stĺpy.

Collider komponenty majú rôzne vlastnosti a nastavenia, ktoré ovplyvňujú spôsob detekcie kolízií. Medzi tieto vlastnosti patrí napríklad možnosť zapnutia/vypnutia fyzickej kolízie, nastavenie kolíznej vrstvy, fyzikálne materiály a ďalšie.

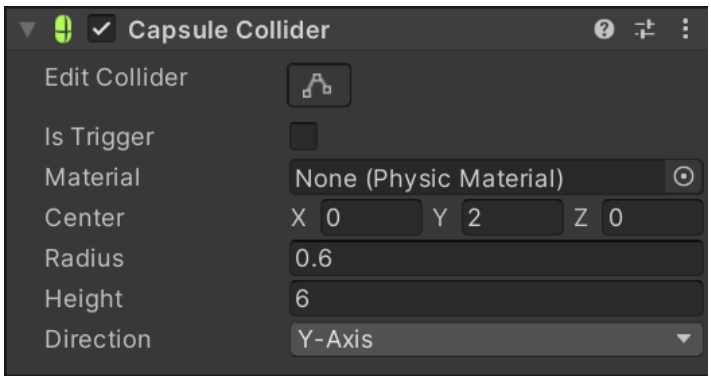

Fig. 6: parametre pre komponent RigidBody

1.1.4 RigidBody

RigidBody je komponent v Unity, ktorý umožňuje simuláciu fyzikálnych vlastností objektov v hernom prostredí. Pomocou RigidBody je možné definovať hmotnosť, gra-

vitáciu, kolízie a pohyb objektu na základe fyzikálnych zákonov. Tento komponent je široko využívaný pre simuláciu pohybu, interakcií a kolízií objektov v 3D priestore.

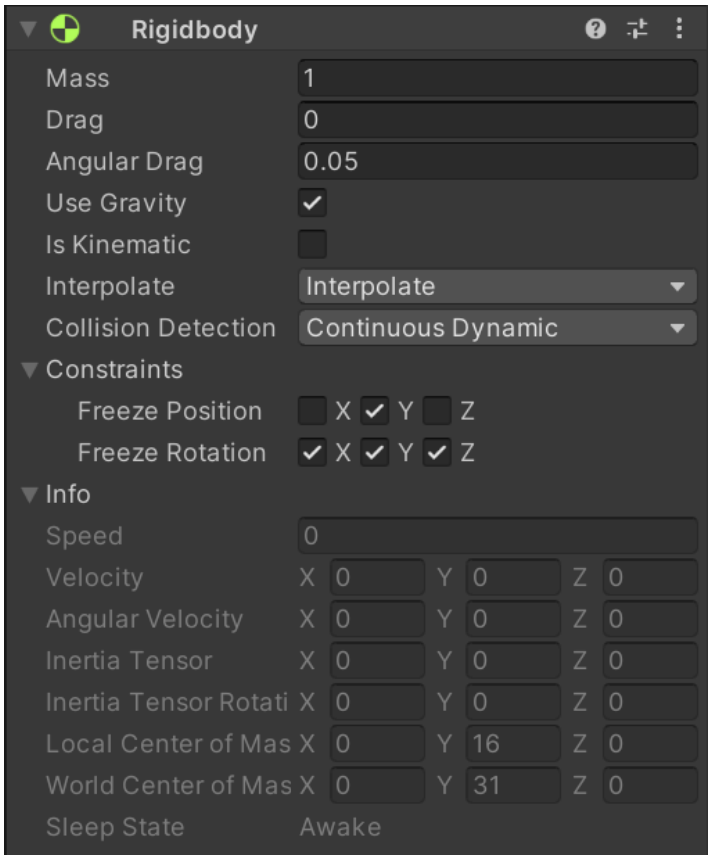

Fig. 7: Transform - uchovávateľ hierarchie

Tieto metódy umožňujú dynamicky ovplyvňovať pohyb a správanie RigidBody v hernom prostredí v závislosti od aplikovanej sily:

- AddForce: Táto metóda umožňuje pridať silu v určenom smere a s intenzitou k RigidBody. Môžete zvoliť medzi rôznymi spôsobmi aplikácie sily, napríklad sila môže byť aplikovaná ako impulz, zrýchlenie, sila.
- AddRelativeForce: Táto metóda umožňuje pridať silu v relatívnom smere k RigidBody. Relatívny smer je založený na rotácii objektu, čo umožňuje pridať silu vzhľadom na jeho smer a orientáciu.
- AddForceAtPosition: Táto metóda umožňuje pridať silu na špecifickú pozíciu na RigidBody. Môžete určiť bod, kde chcete aplikovať silu, a intenzitu a smer sily.
- AddTorque: Ak chcete rotovať RigidBody okolo svojho ťažiska, môžete použiť túto metódu na pridanie torzného momentu.

1.1.5 Scripting - Mono Behaviour

Trieda MonoBehaviour v Unity je základnou triedou umožňujúcou skriptovanie a prispôsobenie správania herných objektov. Slúži ako most medzi hernými objektmi a ich funkcionalitou.

Dedením z triedy MonoBehaviour môžeme vytvárať vlastné skripty, ktoré definujú správanie, interakcie a logiku herných objektov. Tieto skripty možno priradiť k špecifickým herným objektom v Unity editore, čo umožňuje rozšírenie a modifikáciu ich funkcionalít.

Trieda MonoBehaviour poskytuje rôzne metódy a callbacky, ktoré možno prekryť a implementovať konkrétne správanie a reakcie na rôzne udalosti v hre. Niektoré často používané metódy zahŕňajú:

- Start() - Metóda, ktorá sa vykoná pred prvým framom (snímkom).
- Update() - Metóda, ktorá sa vykoná každým framom
- FixedUpdate() - Metóda, ktorá sa vykoná raz za konkrétny časový úsek nezávisle od počtu framov za sekundu, ktoré sú renderované (dôležité pre jednoznačné a jednotné výpočty týkajúce sa fyziky).
- OnTriggerEnter(), OnCollisionEnter() - vykoná sa, keď sa prekryjú dva alebo viaceré collidre.
- GetComponent ⟨DesiredComponentOfGameObject⟩ () - Metóda na získanie komponentov, ktoré sú priradené objektu, ktorému je priradený script (trieda rozširujúca MonoBehaviour).

Celkovo povedané, trieda MonoBehaviour je kľúčovým prvkom v skriptovacej architektúre Unity, ktorý umožňuje vytvárať dynamické a interaktívne správanie herných objektov pomocou písania vlastných skriptov a využitia API Unity.

1.2 História AI v PC hrách

Evolúcia AI vo videohrách predstavuje cestu od jednoduchého k zložitému, zameranú na vytvorenie presvedčivej ilúzie inteligencie namiesto pravého akademického AI. Ranné hry využívali základné techniky AI na tvorbu atraktívnych protivníkov, s hlavným cieľom poskytnúť hráčom zábavný a výzvami nabitý zážitok.

Súčasné AI vo hrách sa vyvinulo nad rámec vytvárania nepriateľov. Teraz zahŕňa vývoj vedľajších postáv a umelých spoluhráčov, čím zvyšuje ponorenie hráča do virtuálneho

sveta. Tento posun zdôrazňuje potrebu, aby NPC prejavovali prirodzené a realistické správanie, prekračujúc zameranie na grafickú vernosť.

Kľúčom k AI vo hrách je koncept "menej je viac", kde jednoduchšie AI často postačuje na vytvorenie pútavého herného zážitku. Typický cyklus AI vo hrách nasleduje trojkrokový proces: vnímať, premýšľať a konať. Tento prístup, hoci priamočiary, sa ukazuje ako účinný v udržiavaní záujmu o hru.

Rozlišovanie medzi AI vo hrách a AI v teórii hier je kľúčové. AI v teórii hier, používané v hrách ako šach, zahŕňa komplexné vyhľadávanie v stromoch a je výpočtovo náročné, čo ho robí menej vhodným pre hry v reálnom čase s mnohými NPC.

Jednou z najväčších výziev v AI pre hry je potreba rozhodovania v reálnom čase. Táto požiadavka často vylučuje určité techniky AI, čo si vyžaduje, aby AI pôsobilo odozvou a rozhodnutia sa javili ako okamžité hráčovi.

Celkovo je história AI vo hrách charakterizovaná postupným prechodom od tvorby protivníkov k zlepšovaniu ponorenia hráča prostredníctvom sofistikovaných, životu podobných NPC, čo odráža širší trend smerujúci k tvorbe presvedčivejších a pútavejších virtuálnych svetov.

1.3 Machinne learning

"Machine Learning"je odvetvie, ktoré sa zaoberá tvorbou počítačových programov, ktoré sa automaticky zlepšujú prostredníctvom skúseností. Táto definícia zdôrazňuje vývoj programov, ktoré sa môžu učiť a prispôsobiť prostredníctvom prijmania a spracovávania dát a skúseností, čo je kľúčovým znakom systémov strojového učenia.[1]

1.3.1 Agent

Agent je nezávislý program alebo entita, ktorá interaguje so svojím prostredím vnímaním svojho okolia pomocou senzorov, potom vykonaním akcie prostredníctvom äktuátoraälebo ëfektora". Agenti používajú svoj äktuátor"na pohyb v cykle vnímanie, spracovanie, akcia.

1.3.2 Reinforcement learning

Učenie Posilňovaním je oblasť umelej inteligencie, ktorá sa vyznačuje výpočtovým prístupom, pri ktorom sa agent snaží maximalizovať odmeny pri interakcii so zložitým a premenlivým prostredím. Toto pole zahŕňa algoritmy učenia, ktoré sa prispôsobujú na základe skúseností agenta, čím sa zdôrazňuje schopnosť agenta robiť rozhodnutia vedúce k najväčšiemu kumulatívnemu prospechu. Je to dynamický proces zameraný

na nepretržité učenie sa a prispôsobovanie sa novým podmienkam, často využívaný v rôznych aplikáciách ako robotika či herný priemysel.[5]

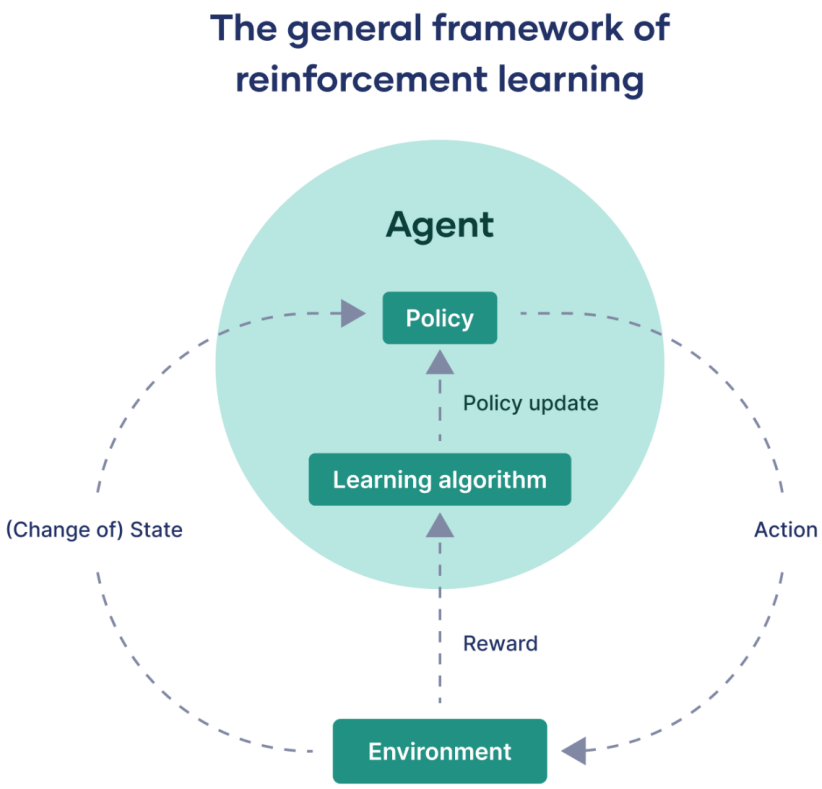

Fig. 8: Reinforcement learning princíp graficky

1.3.3 Proximal policy optimization (PPO)

Proximálna Optimalizácia Politík (PPO) je metóda gradientu politiky pre posilňovacie učenie, ktorá vyvažuje jednoduchosť, ľahkosť implementácie, efektívnosť vzoriek a ľahkosť ladenia. Je známa svojou efektívnosťou v rôznych zložitých prostrediach. PPO rieši problém veľkých aktualizácií politiky, ktoré môžu viesť k zrúteniu výkonu, optimalizáciou "surrogateöbjektívnej funkcie. Táto funkcia zabezpečuje, že aktualizácie politiky nie sú príliš veľké, čím sa udržiava stabilita počas tréningu.

Kľúčové vlastnosti PPO zahŕňajú:

- **Clipped Surrogate Objective**: Mechanizmus strihania na zabránenie príliš veľkým aktualizáciám politiky.
- **Viaceré Epochy Stochastického Gradientového Vzostupu**: Toto aktualizuje politiku viackrát pre každú dávku údajov.

- **Použitie Metódy Actor-Critic**: Používa oddelené siete politík (actor) a hodnoty (critic), hoci môžu zdieľať niektoré vrstvy.[4]

1.4 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) je off-policy algoritmus v posilňovacom učení, ktorý optimalizuje stochastickú politiku öff-policy". Je známy svojou efektívnosťou vzoriek a stabilitou, čo ho robí vhodným pre rôzne náročné úlohy.

Kľúčové vlastnosti SAC zahŕňajú:

- **Framework Actor-Critic**: SAC využíva metódu actor-critic, kde 'actor' aktualizuje politiku a 'critic' odhaduje hodnotovú funkciu.
- **Off-Policy Učenie**: To umožňuje SAC efektívnejšie využiť predtým zozbierané údaje.
- **Regularizácia Entropie**: Táto funkcia podporuje prieskum pridaním entropického členu k odmene, čo pomáha algoritmu širšie skúmať a učiť sa robustnejšie politiky.
- **Nepretržité Akčné Priestory**: SAC je obzvlášť vhodný pre prostredia s nepretržitými akčnými priestormi.

SAC sa ukázal byť efektívnejší ako iné metódy z hľadiska efektívnosti vzoriek a stability učenia, čo ho robí preferovanou voľbou pre mnohé aplikácie v real time hrách a ďalších oblastiach, kde sú vyžadované nepretržité akcie.

1.5 Policy Optimization with Covariance Matrix Adaptation (POCA)

Policy Optimization with Covariance Matrix Adaptation (POCA) je metóda posilňovacieho učenia, ktorá sa zameriava na optimalizáciu politiky s použitím adaptácie kovariančnej matice. Táto metóda kombinuje princípy evolučných algoritmov a gradientových metód posilňovacieho učenia, aby dosiahla efektívnejšiu optimalizáciu politík v komplexných prostrediach.

Kľúčové vlastnosti POCA zahŕňajú:

- **Adaptácia Kovariančnej Matice**: Využíva adaptáciu kovariančnej matice na úpravu smeru a veľkosti krokov v procese učenia, čo umožňuje efektívnejšie preskúmanie priestoru politík.

- **Kombinácia Evolučných a Gradientových Prístupov**: Integrácia týchto dvoch prístupov umožňuje lepšie využitie informácií o gradientoch, zatiaľ čo zároveň udržiava robustnosť typickú pre evolučné algoritmy.
- **Vhodné pre Komplexné Prostredia**: POCA je navrhnutá tak, aby bola účinná v prostrediach s vysokou dimenziou a zložitými dynamikami.[3]

Táto metóda sa ukázala byť sľubná v rôznych aplikáciách posilňovacieho učenia, kde tradičné metódy môžu zlyhať alebo nie sú dostatočne efektívne.

1.6 Unity Machine Learning Agents

Konfigurácia pre tréning Unity ML Agents je špecifikovaná v súbore YAML. Každý parameter v tomto súbore zohráva kľúčovú úlohu v učebnom procese agentov. Nižšie je opis kľúčových parametrov a špecifických nastavení pre algoritmy ako PPO a SAC:

- **behaviors**: Definuje správanie pre rôznych agentov v prostredí.
- **trainer_type**: Určuje algoritmus tréningu, ako PPO alebo SAC.
- **hyperparameters**:
  - **batch_size**: Veľkosť dávky pre spracovanie údajov.
  - **buffer_size**: Veľkosť buffru pre uchovávanie skúseností.
  - **learning_rate**: Rýchlosť učenia modelu.
  - **beta**: Koeficient pre reguláciu (používa sa v SAC).
  - **epsilon**: Parameter pre exploráciu (používa sa v PPO).
  - **num_epoch**: Počet epoch pre tréningovú iteráciu.
- **network_settings**: Nastavenia neurónovej siete, vrátane počtu vrstiev a jednotiek.
- **reward_signals**: Typy odmerných signálov, ako sú extrinzické alebo intrinzické motivácie.
- **max_steps**: Maximálny počet krokov v tréningovej epizóde.
- **time_horizon**: Dĺžka časového horizontu pre odmeny.
- **summary_freq**: Frekvencia, s akou sa generujú súhrnné štatistiky.
- **use_curiosity**: Určuje, či sa má použiť zvedavosť ako odmerný signál.

- **gamma**: Diskontný faktor pre budúce odmeny.
- **normalize**: Určuje, či sa majú normalizovať vstupné údaje.

Tieto parametre a nastavenia sú základom pre konfiguráciu tréningového procesu Unity ML Agents a ovplyvňujú efektivitu a výkon učenia agentov.

2 Implementácia

2.1 Hráč - Warlock

Ako sme už skôr spomenuli objekt môže byť zložený z mnoho podobjektov a môžu mu byť priradené rôzne komponenty. V našom prípade má hráč priradených k objektu viacero skriptov a komponentov. (pozri 2)

Fig. 9: Warlock postava zobrazujúca, čo vidí hráč a aký má collider (CapsuleCollider)

Napriek tomu, že samotná postava je viditeľná ako postava z rukami, nohami, hlavou, žezlom a plášťom, pre potreby fyziky reaguje ako kapsula. V hre sa používajú schopnostim ktoré vytvárajú ďalšie obejkty, ktoré pri kontakte tvoria explózie

a vytvárajú odrážanie podobné biliardu, preto dáva najväčší zmysel používať collidre s kruhovým okrajom. (Pozri fig 9.)

Každý hráč má zobrazené meno a inidkátor života (health bar). Riešenie, ktoré sme využili je, že vrchný obrázok typu "filled"zelenej farby je umiestnený nad spodným obrázkom a jeho "fill amount"sa aktualizuje na základe aktuálneho percentuálneho stavu zdravia. (Pozri Fig 10)

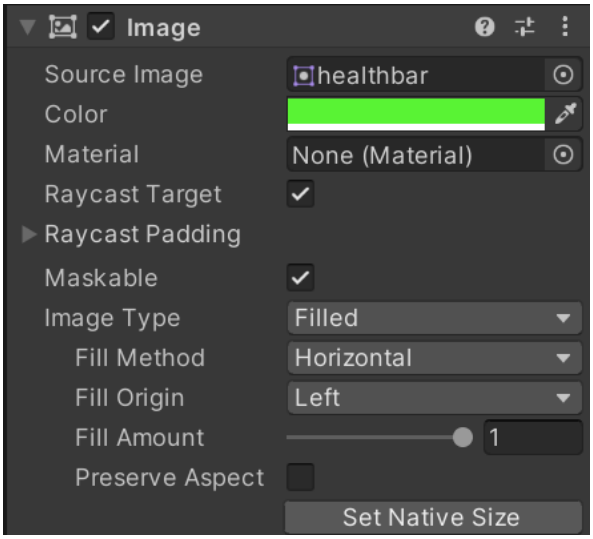

Fig. 10: Health bar obrázok typu "filled"

Taktiež je objektu, ktorý predstavuje healthbar priradený skript, ktorý sa stará o viditeľnosť života a mena hráča tak, že nastavuje svoju rotáciu na indetickú s rotáciou kamery. (Vždy smeruje k sledovateľovi. Pozri Fig 11)

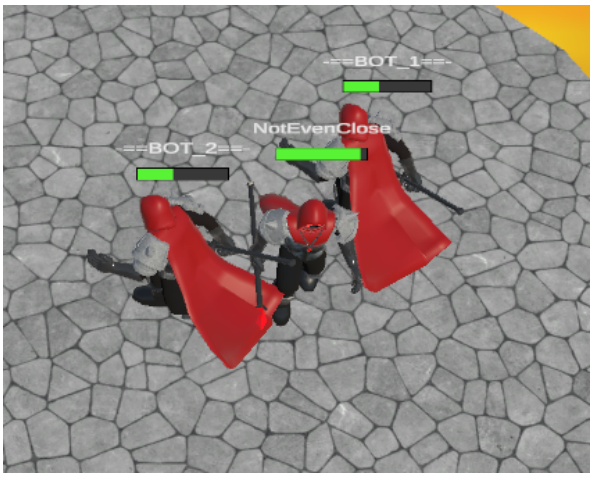

Fig. 11: Health bar natočený smerom ku kamere nezávisle od rotácie objektu "Warlock"

Objekty predstavujúci hráča má funkcionalitu zabezpečenú pomcou týchto skriptov:

Algoritmus 1 PlayerMovement Class

```
1: private rb
2: private movingVector
3: private shouldMove
4: private knockedUpTime
5: procedure Start
6:     rb ← getComponent(Rigidbody)
7:     movingVector ← (0, 0, 0)
8:     shouldMove ← false
9:     knockedUpTime ← 0
10: procedure Update
11:     knockedUpTime ← knockedUpTime − Time.deltaTime
12:     if rightMouseButtonClicked() then
13:         rightClickPosition ← getMouseClickPosGround()
14:         currentPosition ← transform.position
15:         movingVector ← normalize(rightClickPosition − currentPosition)
16:         movingVector.y ← 0
17:         shouldMove ← true
18:     if shouldMove then
19:         if knockedUpTime ≤ 0 then
20:             rb.velocity ← movingVector × movementSpeed
21: procedure FixedUpdate
22:     // Perform physics-related updates
23:     // ...
24: procedure OnFireballHit(knockVector, damageAmount)
25:     knockedUpTime ← knockUpTimeLength
26:     shouldMove ← false
27:     rb.velocity ← (0, 0, 0)
28:     playerStats.TakeDamage(damageAmount)
29:     // ...
30: function GetMouseClickPosGround
31:     // Get the mouse click position on the ground
```

Algoritmus 2 PlayerStats Class

```
1: private currentHealth
2: private healthDrainage
3: procedure Start
4:     currentHealth ← maxHealth
5: procedure Init(id)
6:     _id ← id
7:     _isBot ← id > 0
8:     Enable/disable camera movement and spell screen based on _isBot
9:     Set the names of the player and bots in the healthBarAndName component
10: procedure Update
11:     if not IsWalkingOnSolidGround() then
12:         TakeDamage(5f × Time.deltaTime)
13:     if currentHealth < maxHealth then
14:         HealHp(regenHealth × Time.deltaTime)
15:     TakeDamage(healthDrainage × Time.deltaTime)
16: procedure TakeDamage(damageAmount)
17:     currentHealth ← currentHealth − damageAmount
18:     if currentHealth < 0 then
19:         Destroy(gameObject)
20: procedure HealHp(healAmount)
21:     currentHealth ← currentHealth + healAmount
22:     if currentHealth > maxHealth then
23:         currentHealth ← maxHealth
24: procedure IsWalkingOnSolidGround
25:     Perform a raycast downwards from the player's position
26: procedure InitThrust(thrustData)
27:     _thrustData ← thrustData
28:     isThrusting ← true
29:     thrustLife ← _thrustData.lifeLength
30: procedure OnCollisionEnter(col)
31:     if col.gameObject is a bot then
32:         col.Get(PlayerMovement).OnFireballHit(knockVector, damage)
33:         Reset Thrust
```

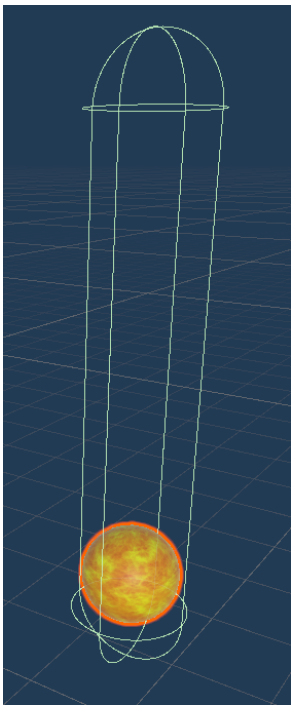

Fig. 12: Vizuálna reprezentácia Fireball a jeho collidera (CapsuleCollider).

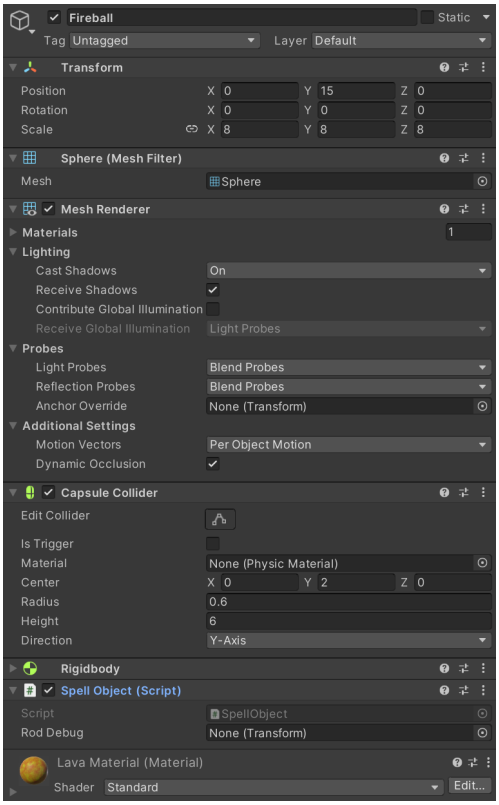

Fig. 13: Komponenty Fireball-u.

2.2 SpellObject

Hráč môže využívať schopnosti, ktoré vytvoria nový objekt, ktorý môže mať rôzne vlastnosti (vybuchne pri kontakte, spôsobí poškodenie, dá hráčovi nejaký status (spomalenie pohybu, omráčenie, nemožnosť používať schopnosti a podobne)). Základnou schopnosťou je ohnivá guľa (Fireball).

2.2.1 Ml agents learning

- **default konfiguračný súbor YAML:**

```
default_settings: null
behaviors:
  My Behavior:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
```

```
    beta: 0.005
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    shared_critic: false
    learning_rate_schedule: linear
    beta_schedule: linear
    epsilon_schedule: linear
  network_settings:
    normalize: false
    hidden_units: 128
    num_layers: 2
    vis_encode_type: simple
    memory: null
    goal_conditioning_type: hyper
    deterministic: false
  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
      network_settings:
        normalize: false
        hidden_units: 128
        num_layers: 2
        vis_encode_type: simple
        memory: null
        goal_conditioning_type: hyper
        deterministic: false
  init_path: null
  keep_checkpoints: 5
  checkpoint_interval: 500000
  max_steps: 500000
  time_horizon: 64
  summary_freq: 50000
  threaded: false
  self_play: null
  behavioral_cloning: null
```

```
env_settings:
  env_path: null
  env_args: null
  base_port: 5005
  num_envs: 1
  num_areas: 1
  seed: -1
  max_lifetime_restarts: 10
  restarts_rate_limit_n: 1
  restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 10.0
  target_frame_rate: -1
  capture_frame_rate: 60
  no_graphics: false
environment_parameters: null
checkpoint_settings:
  run_id: '1'
  initialize_from: null
  load_model: false
  resume: false
  force: true
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

Nasledujúce grafy zobrazujú kumulatívne odmeny, ktoré reflektujú účinnosť nastavení v konfiguračnom súbore počas tréningového procesu.

- **Konfiguračný súbor YAML - SAC model:**

```
default_settings: null
```

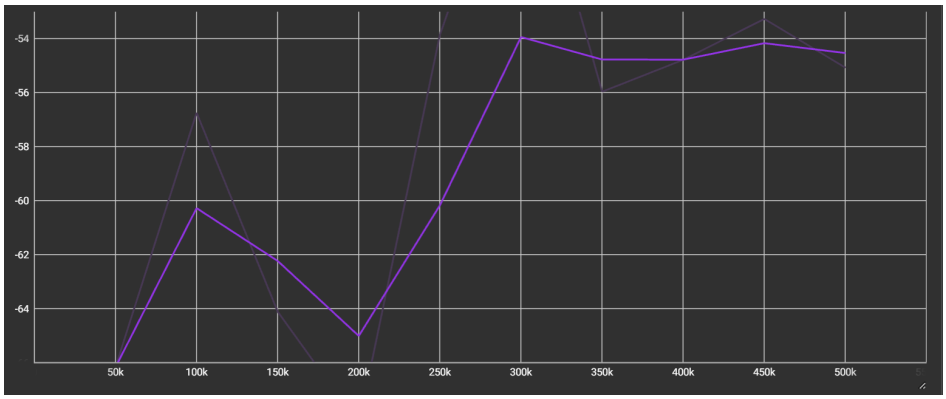

Fig. 14: Cumulative reward using PPO model

```
behaviors:
  My Behavior:
    trainer_type: sac
    hyperparameters:
      batch_size: 256
      buffer_size: 1000000
      buffer_init_steps: 5000
      steps_per_update: 4
      learning_rate: 0.0003
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
    init_path: null
    keep_checkpoints: 5
    checkpoint_interval: 500000
```

```
    max_steps: 500000
    time_horizon: 64
    summary_freq: 50000
    threaded: false
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: null
  env_args: null
  base_port: 5005
  num_envs: 1
  num_areas: 1
  seed: -1
  max_lifetime_restarts: 10
  restarts_rate_limit_n: 1
  restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 10.0
  target_frame_rate: -1
  capture_frame_rate: 60
  no_graphics: false
environment_parameters: null
checkpoint_settings:
  run_id: '1'
  initialize_from: null
  load_model: false
  resume: false
  force: true
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

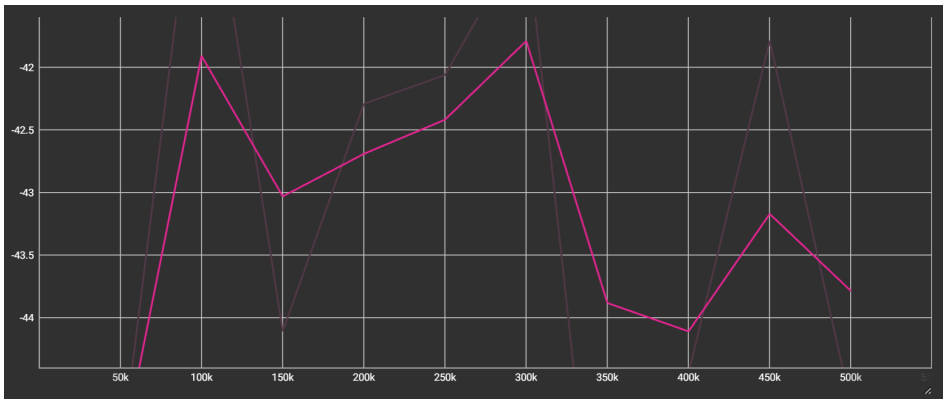

Fig. 15: Cumulative reward using SAC model

- **Konfiguračný súbor YAML - PPO, selfplay:**

```
default_settings: null
behaviors:
  My Behavior:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
```

```
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
    init_path: null
    keep_checkpoints: 5
    checkpoint_interval: 500000
    max_steps: 500000
    time_horizon: 64
    summary_freq: 50000
    threaded: false
    self_play:
      window: 5
      play_against_current_self_ratio: 0.5
      save_steps: 10000
      swap_steps: 2000
    behavioral_cloning: null
env_settings:
  env_path: null
  env_args: null
  base_port: 5005
  num_envs: 1
  num_areas: 1
  seed: -1
  max_lifetime_restarts: 10
  restarts_rate_limit_n: 1
  restarts_rate_limit_period_s: 60
engine_settings:
  width: 84
  height: 84
```

```
  quality_level: 5
  time_scale: 10.0
  target_frame_rate: -1
  capture_frame_rate: 60
  no_graphics: false
environment_parameters: null
checkpoint_settings:
  run_id: '1'
  initialize_from: null
  load_model: false
  resume: false
  force: true
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

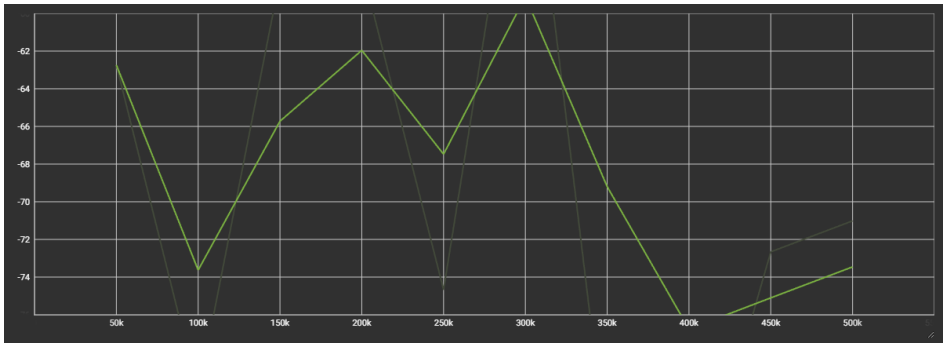

Fig. 16: Cumulative reward using PPO model using selfplay

References

[1] Mitchell, T. M. *Machine Learning.* McGraw-Hill, 1997.

[2] OpenAI. Openai chatgpt. OpenAI API, 2023. Accessed on June 25, 2023.

[3] Qu, X., Gan, W., Song, D., and Zhou, L. Pursuit-evasion game strategy of usv based on deep reinforcement learning in complex multi-obstacle environment. *Ocean Engineering 273* (2023), 114016.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[5] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*, 2 ed. MIT Press, 2018.

[6] Unity Technologies. Unity documentation, 2023.