Silicomrades

Specifikace projektu

Obsah

1. Úvod

2. Přehled herních systémů
    2.1. Herní systémy
        2.1.1. Umělá inteligence
        2.1.2. Spawnery
        2.1.3. Puzzle systém
    2.2. Hráčské postavy
        2.2.1. Input manager
        2.2.2. Movement system
        2.2.3. Weapon system
        2.2.4. Component system
            2.2.4.1. Parts system
            2.2.4.2. Effect manager
        2.2.5. Camera rig
        2.2.6. Health system
    2.3. Managers
        2.3.1. Level manager
        2.3.2. Global game manager
        2.3.3. Game level manager
    2.4. Music manager
    2.5. Animace
    2.6. Uživatelské rozhraní

3. Příběh
    3.1. Příběhové pozadí
        3.1.1. Svět
        3.1.2. Zápletka
    3.2. Epizody
        3.2.1. Epizoda I. - Útěk do Retrolandie
            3.2.1.1. Příběh
            3.2.1.2. Úrovně
        3.2.2. Epizoda II. - Typická návštěva úřadu
            3.2.2.1. Příběh
            3.2.2.2. Úrovně
        3.2.3. Epizoda III.

4. Implementační požadavky
    4.1. [R01] Příběh
    4.2. [R02] Ovládání
    4.3. [R03] Uživatelské Rozhraní
    4.4. [R04] Umělá Inteligence

4.5. [R05] Komponenty a součástky
4.6. [R06] Obtížnost a zamezení snowballingu
4.7. [R07] Hudba
4.8. [R08] Pocit ze hry
4.9. [R08] Efekty
5. Workflow
5.1. Organizace
5.2. Komunikace
5.3. Vedlejší aktivity
5.4. Roadmap
5.4.1. Milestone 1
5.4.2. Milestone 2
5.4.3. Milestone 3
5.5. Rozdělení práce

1. Úvod

V tomto dokumentu se nachází detailní specifikace pro softwarový projekt Silicomrades. Cílem tohoto projektu je pokračovat v práci na herním prototypu vytvořeném během předmětu „Vývoj počítačových her“.

Silicomrades je kooperativní 2D top-down shooter hra pro dva hráče, odehrávající se ve světě plném robotů posedlém kafkovskou byrokracií. Hráči si staví své roboty z různých součástek a poté s nimi plní rozdílné úkoly v epizodických misích. Protože se ovšem jedná o roboty pochybné kvality, tak se během hraní postupně rozpadají a s utrženým zraněním přicházejí o své funkce.

Podrobnější popis výchozího stavu, který tento dokument dále rozvádí, lze nalézt v návrhu projektu na následujícím odkazu: https://docs.google.com/document/d/1rBzBBxWGlb634ISEDk1iKbLnyvQfBBbJsotB2Mlpuxo/edit?ts=5ca73ce1

1.1. Designové cíle

Velkou součástí vývoje počítačové hry je iterace plánovaného designu a posléze testování hypotéz. Je velmi náročné předem o nějaké nápadu s jistotou říci, že bude v praxi skutečně pro hráče zábavný a doplní se skvěle s již existujícími mechanikami. A je to o to náročnější, že spousta designových rozhodnutí v Silicomrades je svým způsobem experiment, u kterého není možné si najít jiné příklady z herního průmyslu, kde takové přístupy reálně fungují.

Designové plány uvedené níže (tj. příběh, mechaniky, dynamiky, nepřátelé, způsoby interakce se světem atd.) jsou tudíž naší aktuální představou toho, co bude pro hru dobré, ale veškeré myšlenky je samozřejmě nutné po implementaci ověřit a v případě, že se neosvědčí, je ze hry odstranit a nalézt vhodnou alternativu, která bude směřovat k námi zamýšlenému zážitku. Pro ulehčení designového procesu jsme si stanovili pár hlavních cílů, jaké je třeba mít vždy při designovém procesu na paměti. Právě cíle uvedené níže pro nás určovaly směr i při návrhu projektu a psaní specifikace.

1.1.1. Téma a atmosféra

Naší cílovou atmosférou je temný byrokratický svět (skoro až kafkovský) s mixem totality, jehož ponurost je narušena absurdnem či jeho vlastní extrémností (všichni berou zákony a byrokracii až příliš vážně).

Svět obývají pouze roboti, kteří jsou hlavní hnací silou byrokratické mašinérie. A mělo by to na nich být patrné. Stejně tak i další grafické elementy hry by měly být v souladu s kýženou atmosférou. Cílem je, aby se dalo u většiny z prvků tvořích atmosféru na první pohled vypozorovat „tohle patří k Silicomrades“.

1.1.2. Konzistence

Všechny aspekty Silicomrades by měly podporovat centrální myšlenku a být tak neoddělitelnou součástí gameplaye, atmosféry či zážitku, jaký se snažíme tvořit. Jednotlivé elementy potřebují dodržovat danou tématiku a nejradši být napojené na již existující systémy - u každého herního elementu by mělo být možné specifikovat, proč ve hře je, jaké naše konkrétní cíle podporuje a s čím je propojen. Systémy, které je možné odejmout (protože nejsou dostatečně propleteny s dalšími herními prvky), by měly být spíše výjimkou.

1.1.3. Kooperativní zážitek

Tvoříme hru pro dva hráče a chceme, aby pro oba aktéry byla sdíleným zážitkem. Zároveň z toho vyplývá, že se snažíme nevytvořit jen prostou zábavu, ale komplexnější zážitek, který dá hráčům něco k prožití (nějaké volby či emoce) a nebo něco k zamyšlení a diskuzi (ať už je to třeba povídání o daném světě a nebo úvaha nad tím, že se na konci sami aktéři stali těmi zlými).

Je důležité se vyvarovat pocitu „dva hráči hrají hru najednou, ale každý sám“, nejradši by Silicomrades bez spolupráce měli být nehratelní či přinejmenším velmi obtížní. S tímto záměrem nám velmi pomáhá i asymetričnost jednotlivých robotů, kdy má každý z hráčů přirozené výhody či nevýhody v jiných situacích a tím pádem se doplňují.

Pocity a reakce, které chceme vzbudit:

- Sounáležitost
- Kamarádství
- Vděčnost
- „Uff, to bylo těsné“
- Radost / smích
- „Výzva, co ji spolu pokoříme“
- Highfive po úspěšně vyhrané úrovni
- Trocha škodolibosti
- Hektická akce
- Obětování se pro druhého

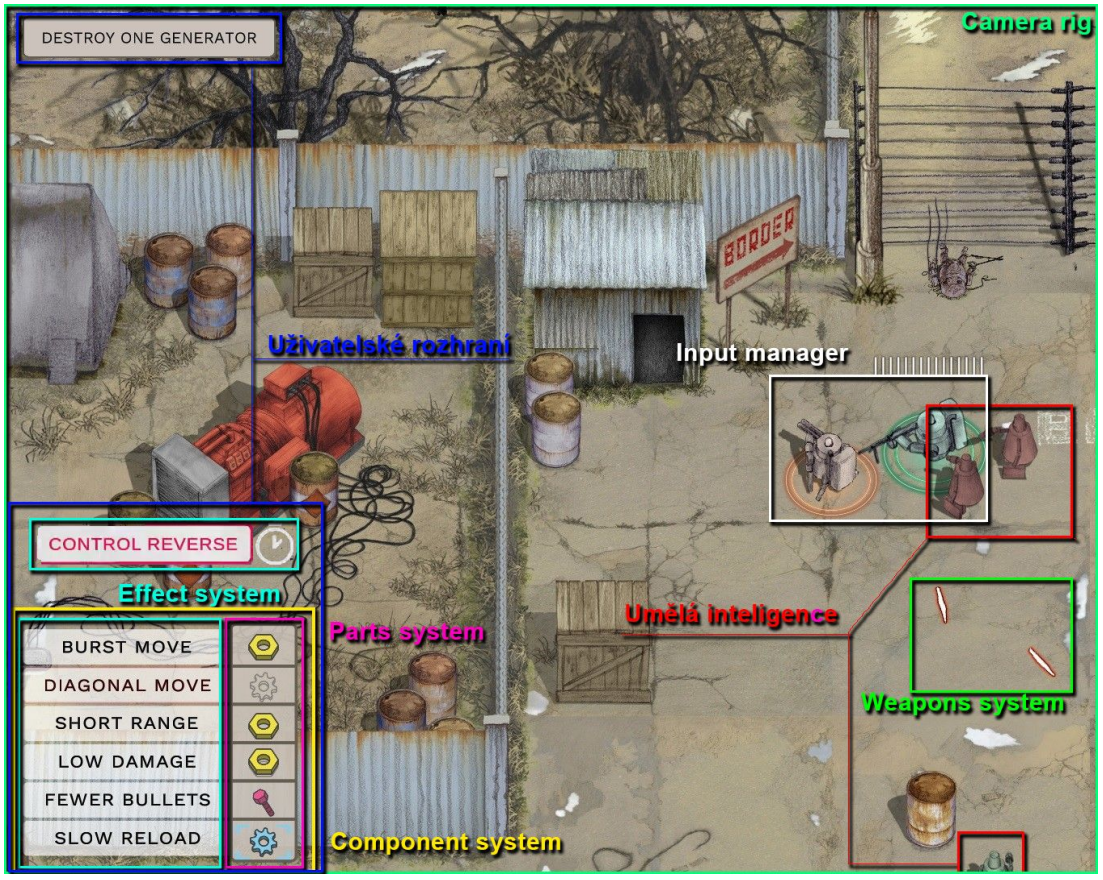
2. Přehled herních systémů

V této části bude popsán přehled hlavních herních systémů. Každý systém představuje logický celek s přesně definovaným účelem. Pro každý z nich bude v této kapitole přehled funkcí a mechanik, jenž má systém na starosti, a způsob, jakým systém pracuje.

Nejedná se o konkrétní přesnou specifikaci požadavků na projekt, nýbrž o snahu přiblížit čtenáři jednotlivé systémy a způsob, jakým pracují. Konkrétní specifikace požadavků je uvedena v kapitole Implementační požadavky. Součástí popisu každého systému je i pod

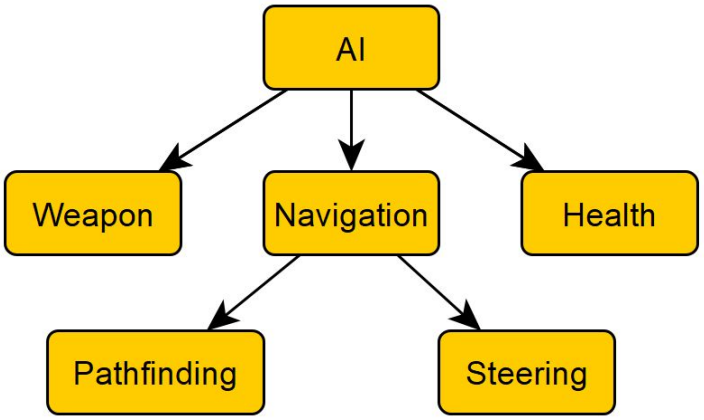
sekce Realizace popisující předpokládanou architekturu, a předběžný zamýšlený návrh způsobu implementace. Vzhledem k použitému hernímu enginu Unity, jsou tyto podkapitoly psány vzhledem k jeho architektuře a za použití obvyklé terminologie s ním spojené.
V Příloze 1. je tato architektura,terminologie, a workflow vývoje při práci s enginem popsána.

Pro architekturu používanou při herním vývoji je typická přítomnost mnoha různých (a často nesouvisejících) systémů obstarávajících jednotlivé funkce a mechaniky. Pro ilustraci je na Obr. 1 zobrazen způsob promítající jednotlivé systémy do výsledného herního zážitku. Jedná se o výřez okna v průběhu hry, kde je vyznačeno, jaké oblasti a informace zajišťuje který ze systémů. Kromě přímo zobrazených systémů na obrázku také v této situaci figuruje systém **Music manageru** (zajišťující správně ozvučení), systém animací, a další interní systémy, které se ve hře neprojeví vizuálně, nýbrž ovlivňují způsob, kterým je se hrou interagováno a jak na tyto interakce reaguje. Každý ze zmíněných systémů je podrobněji popsán v následujícím textu.



Obrázek zobrazuje výřez okna při běžné herní situaci, na kterém jsou vyznačeny některé ze systémů (popsané podrobněji dále) podílejících se na tvorbě herního zážitku.

2.1. Herní systémy



Obrázek zobrazuje, jaké všechny systémy tvoří postavu ovládanou AI

2.1.1. Umělá inteligence

Systém umělé inteligence odpovídá za navigování a chování hráči neovládaných robotů.

Systém zajišťuje:

- Steering behaviors pro navigaci robotů v prostředí
- Výběr nejlepšího cíle pro roboty
- Střelbu a míření robotů
- Speciální schopnosti dle typu robotů

Tento systém zajišťuje veškeré chování nepřátelských robotů. Každý jednotlivý nepřátelský robot na mapě používá tento systém, který mu umožňuje vybírat správným způsobem své chování. Na základě vzdálenosti a viditelnosti hráčů je volena nejlepší strategie a vhodnou navigací terénem a vyhýbání se překážkám v prostředí je provedena. Systém také dle typu robota zajišťuje jeho chování při střetu s hráčem a volí správný cíl pro střelbu či aktivuje další schopnosti robotů.

Roboti se dělí do několika typů:

- Melee enemy
    - Robot bojující zbraní na blízkou vzdálenost, který ve většině případů útočí ve skupině.
- Ranged enemy
    - Robot se zbraní na dálku, který si drží odstup, utíká před hráči, co se přiblíží příliš blízko, a střílí na nejbližšího hráče v náhodných intervalech
- Bomb enemy

    - Rychlý robot, který v momentě, kdy spatří hráče, vyrazí co nejrychleji k němu a exploduje.
- Defect carrier
    - Robot utíkající před hráčem, který v případě, že není do 10 vteřin zničen, způsobí hráčům krátkou globální chybu systémů (=defekt).
- Drone enemy
    - Robot, který se přiblíží k uživatelskému rozhraní, a začne z něj krást součástky, čímž způsobuje hráčům defekty.
- Heavy enemy
    - Robot vyzbrojený útočnou puškou s větší kadencí. Pomalejší a v boji se chová podobně jako ranged enemy.
- Shield enemy
    - Robot vyzbrojený štítem, který ho kryje před veškerým zraněním způsobeným zepředu. Je třeba kooperace obou hráčů, aby ho zneškodnili.

Realizace

Systém zajišťující správný chod umělé inteligence bude rozdělen do několika hlavních součástí, každá zajišťující jednu z navrhovaných funkcí. Během realizace projektu Silicomrades budou implementovány následující komponenty:

Komponenta bezprostřední navigace

Komponenta **Steering behavior** bude implementovat některý z algoritmů používaný k navigaci autonomních agentů v prostředí. Komponenta přijímá za vstup sledovaný cíl, ke kterému se daný objekt chce dostat, další objekty a překážky v bezprostředním okolí kolem objektu, a také set konstant určujících způsob a chování pohybu.

Tyto konstanty určují například váhy jednotlivých Steering behaviors, jejich parametry a maximální velikost sil. Například pro MeleeEnemy bude mít největší váhu Seek behavior s malou nastavenou vzdáleností přiblížení. Bude mít aktivní Separate behavior, aby nedocházelo k přílišnému hromadění nepřátel. Pro RangedEnemy bude naopak mít větší váhu Flee behavior v momentě, kdy se hráč více přiblíží, a vzdálenost přiblížení pro Seek behavior bude velká. Tím bude dosaženo toho, že se RangedEnemies budou držet dále od hráčů.

Na základě těchto vstupů je každý frame informována **Rigidbody2D** komponenta, a je jí předán směr a rychlost, kterou se má daný objekt pohybovat.
Komponenta bude používána pro navigaci k objektům v bezprostřední blízkosti. V případě, že cesta k cílovému objektu není přímočará, bude komunikovat s **GameObjectem Pathfinding manager**, ze kterého získá cestu, kterou bude (za dodržení všech pravidel dle konstant specifikovaných v **Steering behaviors**) následovat. Tím získá prozatimní cíl ve formě nejbližšího vrcholu navigačního grafu, pro který již stačí pouze lokální forma navigace.

Jako kandidáti pro způsob realizace bezprostředního pohybu jsou zatím zvoleny následující algoritmy:

- Steering behaviors/Boids, založeném na algoritmu od Craiga Reynoldse (https://www.red3d.com/cwr/boids/), který je v navigaci (převážně skupin)

autonomních agentů velmi často používán. Kvůli jednoduchosti implementace bude implementován jako první.
- Velocity Obstacle, a jejich Reciprocal a Hybrid Reciprocal varianty (http://gamma.cs.unc.edu/HRVO/). V porovnání se steering behaviors jsou složitější na implementaci, ale v některých případech podávají lepší výsledky.

V případě, že steering behaviors budou podávat nedostatečné výsledky pro naše potřeby, bude přistoupeno k implementaci složitějšího algoritmu HRVO, nebo bude provedena další rešerše a hledání dalších alternativ.

GameObject Pathfinding manager

Tento manager, implementovaný ve formě samostatného **GameObjectu**, bude mít na starost zajišťovat globální pathfinding pro hledání cest většího než bezprostředně lokálního charakteru.
Pro zadaný vstup výchozího a cílového bodu na scéně spočítá nejkratší možnou cestu kudy se k cíli přesunout, aniž by došlo ke kolizi s ostatními collidery na scéně. Doplňuje předchozí komponentu bezprostřední navigace, neboť k navigaci používá znalost o celé mapě, místo pouze raycastů do bezprostředního okolí.

Vlastní **GameObject** pathfindingu při spuštění hry spočítá navigační graf celé úrovně, který si uloží a během hry bude prohledávat za použití algoritmu A*. Generování navigačního grafu proběhne v první iteraci nejspíše způsobem navigační mřížky, převážně z důvodů relativní jednoduchosti implementace. Pokud se navigační mřížka ukáže jako příliš neefektivní pro potřeby projektu, bude implementován některý ze sofistikovanějších algoritmů na generování navigační sítě (navmesh), která zefektivní prohledávání grafu, a umožní rychlejší prohledávání a hledání nejkratších cest.

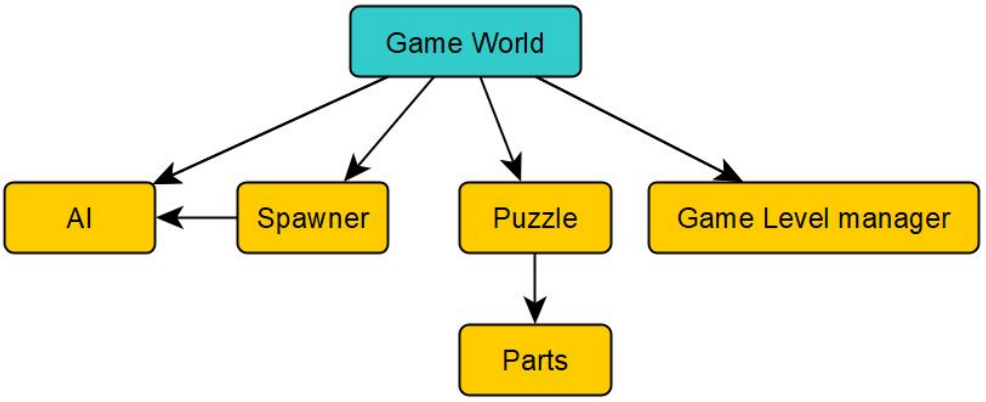
Pathfinding bude implementován jako samostatný globální manager ve formě **GameObjectu** z důvodů cachování, neboť bude pro hru důležité hledat cestu pro desítky objektů každý frame. Objekty se ale budou z velké části nacházet blízko u sebe (neboť půjde o skupiny robotů se stejným cílem), a díky globálnosti a cachování **Pathfinding manageru** bude možné ušetřit velkou část dotazů, které by v případě implementace samostatné komponenty pathfindingu bylo nutné opakovat pro každý objekt zvlášť.

Komponenty speciálních schopností

Každá ze speciálních schopností a chování nepřátel bude implementována formou samostatné komponenty, která v rámci `Update` smyčky zajišťuje potřebnou logiku pro její správný chod a vyhodnocení. Jedná se například o komponentu, která zajišťuje na základě dat od **Collider2D** komponenty (detekující kolize mezi objekty) vyhodnocení správného momentu pro explodování Bomb enemies a v případě, že k němu má dojít, detekuje v okolí **GameObjekty** s komponentou **HealthSystem** (zajišťující zranění a výdrž objektů) a upozorní je na utržené zranění na základě parametrů nastavených v této komponentě schopnosti explodovat. Podobným způsobem budou implementovány i komponenty pro další plánované speciální schopnosti nepřátelských robotů.

Komponenta AI chování

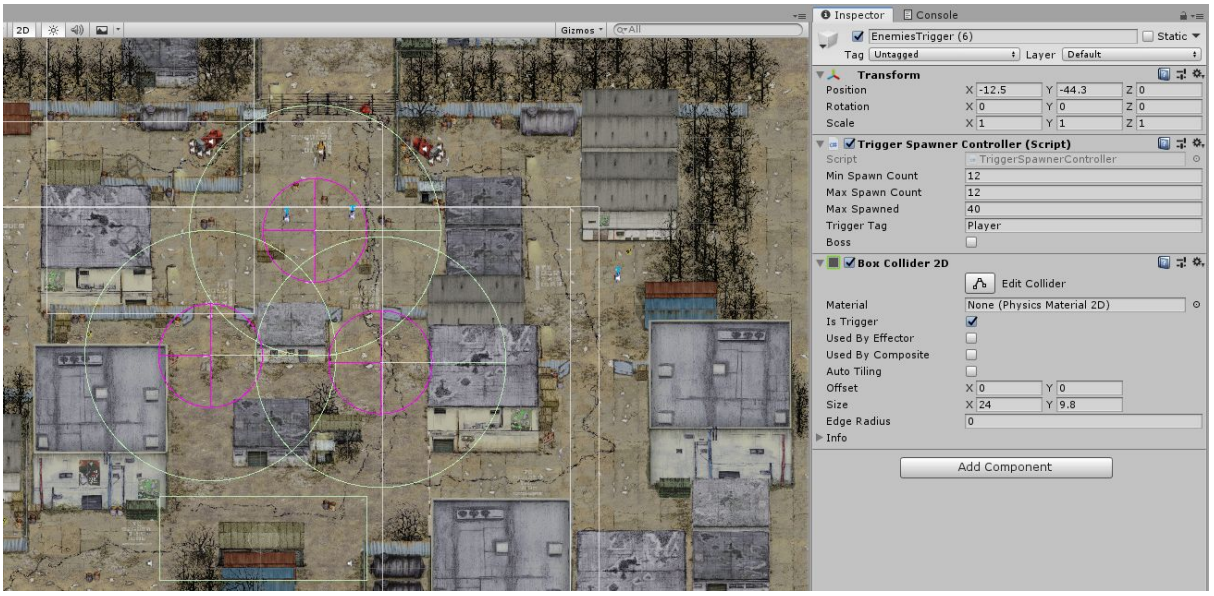
Tato komponenta zajišťuje rozhodování, jakým způsobem a na kterého z hráčů má robot zaměřit svojí pozornost. Dle aktuální pozice **GameObjectu**, na kterém je připojena, pozice hráčů, a konfiguračních parametrů určujících požadované chování (jako například frekvence změny cíle, strategie atd), zvolí komponenta nejvhodnější cíl. K volbě cíle použije modulu Raycasting pro zajištění viditelnosti a po jeho zvolení, a na základě zvolené strategie chování komunikuje s komponentou **Navigace** a komponentou **WeaponSystem**, čímž zajistí vyplnění požadovaného chování.



Obrázek ukazuje, jaké všechny systémy lze nalézt v kombinaci s prostředím světa naší hry. Kupříkladu AI nepřátelé jsou se světem velmi úzce spjati díky nutnosti v něm navigovat.

2.1.2. Spawnery

Systém obstarává aktivaci a deaktivaci nepřátel, aby nedocházelo ke zbytečné simulaci robotů, kteří jsou příliš vzdáleni od hráčů.



Obrázek znázorňuje modelové použití Spawner systému. Po vstupu hráčů do vyznačené oblasti (zelený obdélník) je vytvořen určený počet instancí (v tomto příkladu 12) nepřátelských robotů na zvolených místech (fialové kruhy).

Systém zajišťuje:

- Tvorbu nových instancí nepřátelských robotů na základě triggerů.
- Omezuje maximální počet simulovaných robotů, aby nedocházelo k zahlcení dostupných výpočetních zdrojů.
- Náhodně generuje typy nepřátel při jejich aktivaci na základě zadaných pravděpodobností.

Vzhledem k velikosti plánovaných map bude nutné omezit maximální počet simulovaných robotů. Účelem systému spawnerů bude dát designerům nástroj pro definici míst a způsobu, kterým budou instance nepřátel do hry přidávány, aby nedocházelo ke zbytečné simulaci protivníků, jež jsou příliš daleko od hráčů. Vzhledem k převážně lineární povaze map budou pozice a počty nepřátelských robotů v mapách pevně určený level designerem, a pomocí systému spawnerů budou definovány události, které mají aktivovat tyto předem rozestavěné instance nepřátel.

Spawnery budou podporovat následující události:

- Trigger - Spawner se spustí v momentě, kdy alespoň jeden z hráčů projde předem definovanou zónou.
- Timed - Spawner se aktivuje v pevně daných časových intervalech. Může se aktivovat vícekrát.

- Wave - Spawner je aktivován nějakým jiným objektem - skriptem nebo jinou jednorázovou událostí.

V momentě, kdy je spawner aktivován, vytvoří předem stanovený počet instancí nepřátel, které aktivuje a vypustí do světa. Složení typů nepřátel je náhodně generováno na základě procentuálních šancí, které jsou parametrem každého spawneru. Každý spawner bude obsahovat set předem určených lokací, na které může instance nepřátel vytvořit, ze kterého vybere nejvhodnější pozici. Pozici vybírá takovým způsobem, aby nebyl v momentě aktivace nepřátelský robot na obrazovce nebo blízko hráčů, aby nebyla narušena iluze toho, že robot byl na místě celou dobu.

Realizace

Systém spawnerů bude realizován formou **GameObjectů**, které reagují na eventy typu `OnTriggerEnter` zajišťované Unity komponentou **Collider2D** a na základě nastavených parametrů chování jednotlivých spawnerů zajistí přidání **GameObjectů** požadovaných nepřátelských robotů do scény.

GameObject spawner controller

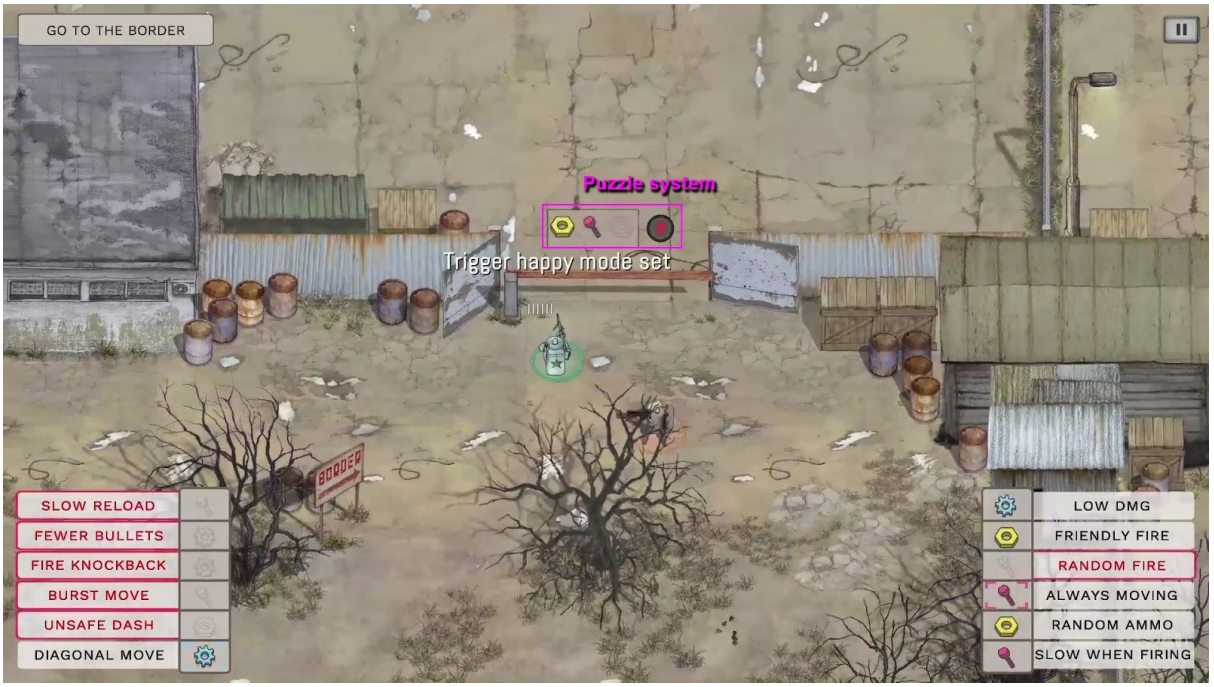
Tento **GameObject** bude zajišťovat management **Spawner GameObjectů**, které k němu budou přiřazeny. Dle parametrů určujících způsob, kterým se objekt chová (zda se jedná o Time, Trigger nebo Wave spawner, kolik nepřátel má po svém spuštění instanciovat atd.), komunikuje s k sobě přiřazenými **Spawner GameObjecty** a notifikuje je o tom, kdy a kolik nepřátel mají instanciovat.
Pro typy spawnerů reagujících na nějaké události (jako například Trigger typ spawneru) používá **GameObject** eventy relevantních komponent (jako například **Collider2D**) a na jejich základě notifikuje své přiřazené **Spawnery**.

GameObject Spawner

Tento **GameObject** zajišťuje logiku instancování nových nepřátel. Nadřazený **GameObejct Spawner controller** notifikuje **Spawner** o počtu nepřátel, které je požadováno instanciovat a na základě svých parametrů **Spawner** náhodně vybere typy nepřátel a v případě, že se v okolí nenachází žádní hráči (a tudíž není zobrazen na obrazovce), nepřátele instanciuje za použití `GameObject.Instantiate()`.

2.1.3. Puzzle systém

Úkolem puzzle systému bude zajistit interakce hráčů s prostředím, s ním spojené uživatelské rozhraní a aktivaci ostatních systémů, událostí a skriptů.



Obrázek zobrazuje puzzle systém ve formě závory, která výměnou za součástku zpřístupní alternativní cestu.

Systém obstarává:

- Interakce hráčů s objekty v prostředí a s ním spojené uživatelské rozhraní.
- Poskytuje rozhraní umožňující designerům jednoduše vyvolávat události a spouštět skripty na základě interakce hráčů s prostředím.
- Komunikace s parts systémem (systémem součástek) a zajištění správného odebrání součástek v případě, že to interakce s prostředím vyžaduje.

Ve hře je plánováno několik herních mechanik, které vyžadují od hráčů odevzdání součástky z inventáře za účelem ovlivnění nějakého objektu. Jedná se například o opravu dveří výměnou za součástku, zprovoznění pastí, či vypnutí alarmu. Úkolem tohoto systému je zajistit tuto interakci, poskytnout hráčům uživatelské rozhraní s ní spojenou a spustit požadované události a skripty v momentě, kdy interakce proběhne.

Systém bude využit minimálně v následujících případech:

- Otevření alternativní cesty výměnou za součástku
- Prodloužení časového limitu na dokončení úrovně výměnou za součástku
- Snížení úrovně alarmu výměnou za součástku
- Udělení velkého množství zranění bossovi výměnou za součástku
- Aktivace pastí nacházejících se v prostředí
- Aktivace turretů, které spustí palbu na nepřátelské roboty

Realizace

Puzzle systém bude realizován implementací kombinace **GameObjectu**, zajišťujícího zobrazení a správu relevantního uživatelského rozhraní, a komponenty **Puzzle**, která

nabídne rozhraní pro interakci s **Parts systémem** a také možnost za použití delegátů spouštět ostatní skripty a komponenty v případě, že je puzzle naplněn součástkami.

GameObject PuzzleUI

Tento **GameObject** bude obsahovat komponentu **PartsUI** a obrázky a objekty nutné pro správně zobrazení uživatelského rozhraní týkajícího se puzzle systému. Také zajišťuje jeho zobrazení a skrytí v případě, že se hráči přiblíží. K tomu používá eventy `OnTriggerEnter/Exit` od **Collider2D** komponenty. V případě, že je **Puzzle** plně naplněn součástkami, dojde ke kontaktování **Puzzle** komponenty, která spustí relevantní skripty na základě funkcí, které se k ní přihlásili. K tomu využívá delegáty a UnityEvent systém.

Komponenta Puzzle

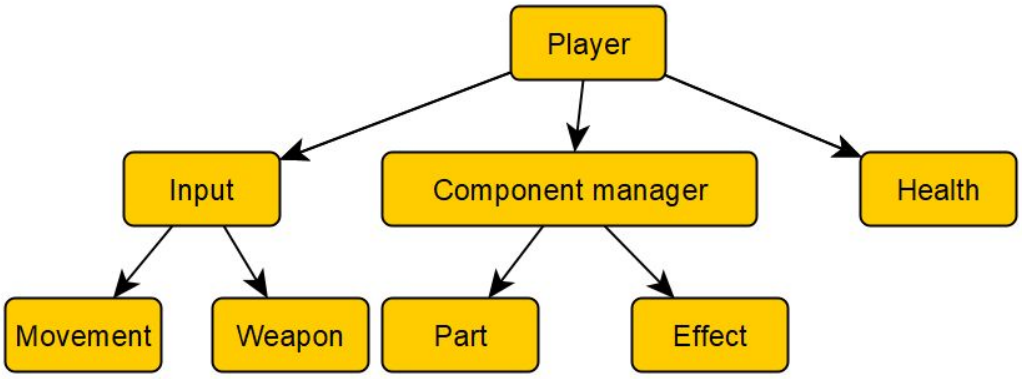
Tato komponenta je sočástí **GameObjectů** implementujících různé výše zmíněné puzzly. Za použití UnityEvent modulu dovoluje designerům implementovat různé druhy puzzlů. Bude poskytovat `OnPuzzleFilled` event, který bude použit dalšími komponentami pro spuštění skriptů a akcí v momentě, kdy je puzzle naplněn.

GameObjecty jednotlivých puzzlů

Každý z výše zmíněných plánovaných puzzlů, bude implementován jako samostatný **GameObject**, který obsahuje svoji instanci **PuzzleUI**, komponentu **Puzzle**, která nabízí event OnPuzzleFilled, a další komponenty jednotlivých puzzlů, které zajišťují jejich správný chod a logické chování.

Každý z typů puzzlů bude mít implementovanou vlastní **Controller komponentu**. Pro puzzly typu Turret to například bude kombinace komponent **WeaponSystem**, komponent **AI chování**, a námi implementované komponenty **TurretPuzzleController**, která reaguje na `OnPuzzleFilled` event, a aktivuje a správně nastaví připojené komponenty. Jako další příklad lze uvést třeba puzzle závory, kterého komponenta **ZavoraPuzzleController** reaguje na event OnPuzzleFilled vypnutím příslušné **Collider2D** komponenty, a spuštěním animace otevření závory, čímž uvolní vstup.

2.2. Hráčské postavy



Obrázek zobrazuje, jaké všechny systémy jsou zapotřebí pro zajištění kompletní funkcionality hráčské postavy.

2.2.1. Input manager

Tento systém bude sloužit pro propojení vstupů od hráčů s ostatními systémy, které zajišťují ovládání a chování dvou hlavních postav - Crawlbota a Jetbota.

Systém má na starost:

- Mapování akcí robotů dle zvoleného ovládacího schéma na použité herní periferie.
- Zpracování uživatelských vstupů a jejich přeposílání ostatním systémům dle definovaného mapování.
- Management připojených periferií a jejich přiřazení hráčům.

Účelem tohoto systému bude oddělení logiky handlingu vstupů od akcí prováděných roboty, aby nedocházelo ke zbytečnému couplingu. Díky **Input manageru** bude možné měnit ovládací schéma nebo použité periferie (klávesnice a myš nebo gamepad) aniž by bylo nutné zasahovat do ostatních systémů robotů.

Realizace

Jedním z velkých nedostatků Unity je problém nedostatečné univerzálnosti detekce vstupů. Unity podporuje pouze obecné mapování akcí na klávesy nebo osy jednotlivých joysticků, což je v praxi nepoužitelné vzhledem ke způsobu, jakým různé systémy pracují s různými druhy periferií. Použití základního **Unity Input manageru** by znamenalo namapovat stovky různých kombinací pro různé periferie, a složitě detekovat jaká periferie je připojena do jakého portu. Z toho důvodu bude použita některá z externích knihoven, která tuto funkcionalitu nabízí, a zprostředkovává mapování různých druhů periferií na definované akce.

Komponenta InputHandler

Knihovna poskytuje možnost detekovat události typu `OnButtonDown`, či směry os joysticků. Námi implementovaná komponenta **InputHandler** bude reagovat na tyto druhy eventů a na základě vstupu od uživatele komunikovat s ostatními systémy a komponentami. Bude delegovat tyto události dál, aby na jejich základě provedla správná komponenta požadovanou akci. Příkladem může být například předání směru vstupu od joysticku systému **MovementSystem**, aby mohl hráče posunout správným směrem.

2.2.2. Movement system

Systém zajišťuje pohyb hráčova robota. Jeho orientaci, rychlost a schopnosti, které se přímo týkají způsobu, jakým se pohybuje v herním světě. Chování systému je určeno typem robota, na kterého se váže.

Systém odpovídá za:

- Způsob pohybu hráčova robota, podle jeho typu
- Schopnosti, které se týkají pohybu
    - Crawlbotův kotoul
- Parametry ovlivňující pohyb robotů, vycházející například z defektů
    - Rychlost
    - Omezení směrů pohybu
- Omezuje vzdálenost, na jakou se od sebe mohou hráči vzdálit

Každý z hratelných robotů má jiný způsob pohybu, aby bylo dosaženo asymetričnosti, která je jedním z našich hlavních design cílů. V případě, že se roboti příliš vzdálí od sebe, systém zajistí, aby nemohli pokračovat dále. A to tím, že je oba lehce odrazí směrem ke středu obrazovky.

Crawl Robot

- Pohybuje se pouze v osmi směrech.
- Vždy míří tím směrem, kterým se pohybuje.
- Schopnost rychlého kotoulu.
- Jako vstup dostává směr, kterým se má pohybovat.

Jet Robot

- Nemá omezení na směr pohybu.
- Může mířit jiným směrem, než kterým se pohybuje.
- Jako vstup dostává směr, kterým se má pohybovat a jiný směr, kterým má mířit.

Realizace

Vzhledem k tomu, jak moc se liší způsob pohybu Crawl a Jet robota, bude jejich pohyb implementován jako dvojice podobných komponent **JetMovementSystem** a **CrawlMovementSystem**.

Komponenta JetMovementSystem

Komponenta na základě vstupů od **InputHandleru** (ve formě směrového vektoru určujícího požadovaný směr pohybu a směr míření) informuje komponentu **Transform**, aby otočila **GameObject** robota do správného směru. Informuje také komponentu **Rigidbody2D**, a předá jí směr a rychlost pohybu. Samotná komponenta nebude pouze předávat informace mezi systémy, nýbrž zajistí jejich zpracování na základě aktivních efektů, defektů, a parametrů pohybu, jako je například rychlost, akcelerace a nebo citlivost na vstupy.

Komponenta CrawlMovementSystem

Narozdíl od Jetbota je pohyb Crawlbota limitován do osmi směrů. Z toho důvodu je nutné rozdělení **MovementSystem** komponent na dvoje odlišné komponenty. Vzhledem k tomu, že je způsob zpracovávání vstupu od **InputHandleru** tolik rozdílný, nedochází k duplikaci kódu i přes to, že komponenty působí vcelku podobně. **CrawlMovementSystem** zajišťuje logiku potřebnou pro převedení vstupu, ve formě vektoru směru požadovaného pohybu od **InputHandleru**, na nejbližší z osmi směrů, jeho naškálování a upravení na základě aktivních efektů, a předání **Rigidbody2D** komponentě jako rychlostní vektor pohybu. Také zajišťuje logiku schopnosti kotrmelce.V reakci na uživatelský vstup a požadovaný směr kotrmelce informuje komponenty **Animatoru**, **Rigidbody2D**, **HealthSystem** a ostatní relevantní komponenty o provedené akci a organizuje správné načasování a průběh této schopnosti.

2.2.3. Weapon system

Systém zajišťuje funkce týkající se zbraní a střelby.

Systém má na starosti:

- Logiku střelby a instancing nových projektilů
- Ofenzivní schopnosti robotů
    - Knockback schopnost Jet robota
- Parametry zbraně robota
    - Počet nábojů v zásobníku
    - Doba přebíjení
    - Kadence střelby
    - Druh vystřeleného projektilu
    - Udělené poškození
    - Místo, ze kterého je projektil vystřelen vzhledem k modelu

Zbraňový systém obsluhuje veškerou logiku týkající se ofenzivních schopností robotů. Omezuje a realizuje střelbu na základě vstupů od **Input manageru** a zbraně, kterou je robot vybaven. Ve hře budou minimálně následující typy zbraní:

- Pistole
    - Semi-automatická střelba
    - Střední počet nábojů v zásobníku
    - Středně dlouhá doba na přebití
    - Průměrné poškození

- Kulomet
    - Automatická střelba
    - Vysoký počet nábojů v zásobníku
    - Středně dlouhá doba na přebití
    - Malé poškození
- Brokovnice
    - Nutné přebití po každém výstřelu
    - Krátká doba na přebití
    - Velké množství projektilů vystřelených najednou

Systém má na starosti také instancing projektilů v případě, že dojde ke střelbě. Zvolí správný počet a typ projektilu na základě zbraně, vytvoří nový projektil a vyšle ho správným směrem dle místa, na které hráč míří. Dále zajistí, aby bylo možné střílet pouze tak často, jak to dovolí kadence zbraně. Udržuje si aktuální počet nábojů v zásobníku a v případě, že dojdou, vypnutí a obstará logiku přebití.

Realizace

Zbraňové komponenty budou realizovány za využití polymorfismu, se základní abstraktní třídou **BaseWeapon**. Třída **BaseWeapon** bude dědit ze třídy **MonoBehavior**, čímž zajistí, že všichni její potomci mohou být použiti jako komponenty.

BaseWeapon

Abstraktní **BaseWeapon** nabídne základní rozhraní nutné pro zbraně a zajistí logiku pro společnou funkcionalitu, jako je instancing projektilů na dané místo s daným směrem, či funkce Shoot(), která poskytne možnost ostatním systémům ze zbraně vystřelit.

Komponenty konkrétních zbraní

Každá ze zbraní bude implementována vlastní komponentou, dědící z třídy **BaseWeapon**. Každá zbraň se liší konkrétním způsobem přístupu k nabíjení, střelbě, či počtu, směru a typu projektilů, které při střelbě instancuje. Jednotlivé komponenty obstarají tuto logiku, a implementují vlastní metodu `Shoot()`, která na základě daných parametrů a navrženého způsobu chování instanciuje správný počet projektilů.

GameObject projektilů

GameObject projektilu disponuje **ProjectileController** komponentou se základní logikou. V momentě, kdy je instanciován, jsou mu od **BaseWeapon** předány parametry jeho chování - převážně směr pohybu a udělené zranění, a projektil předá směr komponentě **RigidBody2D**. V momentě, kdy obdrží **ProjectileController** od komponenty **Collider2D** detekovanou kolizi, zajistí potřebnou logiku pro udělení zranění trefenému objektu, přehrání správných efektů, a smazání instance projektilu.

2.2.4. Component system

Tato kolekce systémů zajišťuje hlavní mechaniku hry Silicomrades - součástky a komponenty ze kterých jsou roboti postaveni a o které přichází, což se projevuje skrze efekty a defekty, které má na starosti effect manager.

Každý z hráčů si vybere několik komponent. Ke každé komponentě se váže nějaký bonus ve chvíli, kdy je naplněna součástkou, a defekt v případě, že součástka chybí.

2.2.4.1. Parts system

Systém zajišťuje správu a manipulaci se součástkami.

Má na starosti:

- Inventář součástek robota a jeho stavu
    - Která komponenta vyžaduje součástku jakého typu
    - Která komponenta je naplněna součástkou
- Manipulaci se součástkami, jejich přesouvání mezi komponenty
    - Hráči si mohou sami sobě i navzájem opravovat komponenty stejného typu pomocí přesunu součástek
- Interakci s prostředím, které vyžaduje obětování součástky
- Přidávání a odebírání součástek z inventáře robota
- Komunikace s effect managerem na základě změny ve stavu komponent

Hráči začínají s plným inventářem, kde každá komponenta má svoji součástku. V průběhu hry o součástky přicházejí utrženým zraněním, nebo tím, že součástku vymění (ať už mezi sebou, nebo s prostředím). Úlohou tohoto systému je provádět tyto změny, udržovat aktuální stav komponent robota, a komunikovat s effect managerem o bonusech a defektech, které má na základě těchto změn robot obdržet. Systém poskytuje následující akce pro manipulaci se součástkami:

- Předání součástky do rozbité komponenty druhého hráče
- Předání součástky do prostředí
- Přidání náhodné nové součástky do rozbité komponenty a její opravení
- Přesunutí součástky z jedné komponenty do jiné
- Odebrání součástky z komponenty (například utržením zranění)

Realizace

Parts systém bude realizován komponentou **PartsSystem**, která si udržuje momentální stav inventáře, informuje komponentu a **GameObjekty** uživatelského rozhraní o změnách, a komunikuje s **EffectManagerem** o defektech a efektech spojených s aktivními a rozbitými součástkami.

2.2.4.2. Effect manager

Systém zajišťuje efekty, které modifikují parametry a chování hry, kamery a robotů. Odpovídá za:

- Seznam možných defektů a bonusů, které může robot daného typu dostat
- Seznam aktuálně aktivních defektů a bonusů daného robota
- Modifikace aktuálního stavu bonusů a defektů robota na základě vnějších událostí
- Management jednotlivých aktivních defektů a bonusů a jejich promítnutí do hry
    - Provedení správné logiky v momentě, kdy je daný efekt aktivován, během toho, co je aktivní, a když končí

Na základě zvolených komponent a jejich aktuálního stavu má každý robot aktivní nějaký set bonusů a defektů, které ovlivňují jeho chování. Úkolem **Effect manageru** je zajistit logiku spojenou s těmito efekty, jejich aplikaci, průběh a ukončení. Také reaguje na změny v aktivních a rozbitých komponentách a podle nich mění set aktivních bonusů a defektů. Změna aktivního stavu efektů se ve většině případů projevuje změnou nějakého parametru (například zpomalení robota, snížení kadence střelby, zákaz ne-diagonálního pohybu atd.), v některých případech je ale potřeba provádět logiku efektu každý frame. Povinností effect manageru je dohlížet na správné obnovení ovlivněných parametrů na původní hodnotu ve chvíli, kdy byl efekt deaktivován.

Effect manager reaguje na tyto události:

- V případě, že komponenta robota přijde o součástku, aktivuje se defekt spojený s komponentou a deaktivuje se bonus.
- V případě, že je komponenta opravena, deaktivuje se defekt a aktivuje se bonus.
- V případě, že nepřítel typu Defect carrier způsobí globální defekt, effect manager ho aplikuje.
    - Po uplynutí stanoveného času, po který má globální defekt působit, jej effect manager opět deaktivuje.

Realizace

Oproti výchozímu stavu projektu, ve kterém je **Effect manager** realizován jednou Singleton třídou pro každého robota, dojde jako součást projektu k návrhu nové, udržitelnější a jednodušeji rozšiřitelné architektury. Dochází k němu z důvodu, že přidávání nových efektů a defektů je jedním z hlavních způsobů rozšíření hry, a do budoucna jsou plánovány rozličné mechaniky, které bude nutné na effekt a defekt systém napojit.

Komponenty Efektů

Plánovaný způsob implementace počítá s refaktoringem logiky jednotlivých efektů do samostatných komponent, dědících třídu **BaseEffect**, jejichž stav a životní cyklus Aktivace -> Běh -> Deaktivace bude spravován komponentou **Effect manager**. Každá z komponent bude tedy přítomna v deaktivovaném stavu na objektu robota již od jeho instancingu, díky čemuž bude možné logiku, parametry a vazby potřebné pro jednotlivé efekty napojit přímo na ně, místo toho aby se nacházeli na skriptech. Například tedy místo toho, aby se komponenta **MovementSystem** dotazovala Singletonu **Effect manageru**, zda není aktivní defekt zpomalení, a tudíž má být snížena rychlost, bude efekt zpomalení reprezentován komponentou zpomalení, která v momentě, kdy je aktivována, notifikuje **MovementSystem**, že došlo ke zpomalení hráče (za použití níže zmíněné modifiable variable). V o něco méně triviálním případě může jít například o komponentu efektu rozbité obrazovky, který ve svém Update skriptu po své aktivaci zajišťuje logiku třesení a blikání obrazovky.

Modifiable variable pattern

Dalším z plánovaných způsobů refaktoringu je implementace datového typu modifikovatelné proměnné, která umožní ostatním třídám modifikovat hodnoty ostatních parametrů, aniž by s nimi musela přímo komunikovat třída, která proměnnou vlastní. Modifikovatelná proměnná bude disponovat metodou GetValue, která aplikuje na základní hodnotu (kterou určuje třída,

pod kterou spadá) veškeré modifikace, které ostatní třídy a komponenty přihlásili. Jako příklad lze uvést parametr rychlost u komponenty **MovementSystem**. V případě, že bude efekt zpomalení chtít tento parametr, který je instancí modifikovatelné hodnoty, snížit na polovinu, použije metodu modifikovatelné proměnné, a přidá do seznamu aktivních modifikací násobení hodnotou 0.5.
Tímto způsobem bude možné jednoduše modifikovat a ovlivňovat parametry a hodnoty, aniž by bylo potřeba příliš velkého couplingu, nebo nutnosti příliš provazovat systémy dohromady.

2.2.5. Camera rig

Systém camera rig zajišťuje pohyb kamery a poskytuje rozhraní pro její ovládání v případech, kdy je od ní pro potřeby některých úrovní vyžadováno jiné než základní chování.

Systém zajišťuje:

- Sledování dvojice hráčských postav kamerou, její pohyb a přiblížení/oddálení na základě vzdálenosti jednotlivých robotů takovým způsobem, aby v každém případě byl zobrazen dostatek herní plochy pro pohodlnou hratelnost.
- Poskytuje rozhraní umožňující jednoduše na základě událostí ve hře zaměřit pozornost kamery na jiné objekty dle potřeb level designu.
- Vyhlazování pohybu kamery, aby vypadal přirozeně a nebyl trhaný.

Kamerový systém obstarává pohyb kamery a reaguje na události s kamerou spojené. Hlavní zodpovědností systému je udržovat hráčské roboty na obrazovce vhodným pohybem, přiblížením a oddálením kamery. Požadované chování kamery při sledování hráčů je následující:

- Pohyb a přiblížení kamery je interpolováno a tlumeno, aby nebylo rušivé.
- Střed kamery je co nejblíže středu vzdáleností mezi hráči.
- Kamera je přiblížena takovým způsobem, aby zachovala mezi hráči a krajem obrazovky malý rozestup z důvodu větší přehlednosti.
- V případě, že hráč míří směrem mimo obrazovku, mírně kameru posunout daným směrem.
- Pokud se od sebe hráči příliš vzdálí, postupně ztmavit obrazovku, dokud se nepřiblíží zpět (maximální vzdálenost hráčů je kontrolována **Movement systémem**, kamera reaguje pouze vizuálně).
- Zobrazení vizuálních defektů.

Další zodpovědností kamery je umožnit designerům definovat události, při kterých kamera opustí základní chování, a přesune se na určené místo. Je požadováno zvládnutí těchto situací:

- Sledovat více objektů, než jen hráče.
    - Střed kamery se drží co nejblíže těžišti mezi sledovanými objekty a hráči.
    - Kamera je přiblížena způsobem, aby obsáhla hráče a požadované objekty.
    - Toto chování lze aktivovat a deaktivovat skriptem, či v reakci na událost ve hře (jako například příchod hráčů na určité místo).

- Přiblížit se na daný objekt
    - Kamera se hladce přiblíží na zvolený objekt.
- Třesení kamery, např. v případě exploze.

Poslední zodpovědností kamery je zobrazit kamerové defekty. Kamerový systém zprostředkovává tyto defekty:

- Camera glitch - na obrazovce se vyskytují barevné artefakty, podobné artefaktům které způsobí špatný signál CRT televizím
- Camera shake - kamera se v daných intervalech silně zatřese

Realizace

Kamerový systém je realizován za použití Unity komponenty **Camera**, která obstarává veškerou logiku potřebnou pro nastavení, limitování a vykreslení viewportu na základě zadaných parametrů, jako je například velikost, poměr stran obrazovky, či požadované zobrazení. Námi je implementována komponenta **CameraRig**, která zajistí pohyb a přiblížení kamery (prováděné změnou velikosti) na základě požadovaného chování, popsaného výše.

Komponenta CameraRig

Komponenta je realizována jedním skriptem. **CameraRig** dostane jako vstup seznam referencí na objekty, které má udržovat na obrazovce, stejně jako další parametry limitující její chování (rychlost přiblížení, faktor vyhlazování pohybu, maximální a minimální přiblížení, minimální vzdálenost sledovaných objektů od kraje obrazovky atd), a na jejich základě přesune a přenastaví parametry **GameObjectu** kamery a jeho **Camera** komponenty. Výpočet velikosti a pozice kamery je triviálním problémem analytické geometrie.

2.2.6. Health system

Úlohou tohoto systému je zajistit správné interakce projektilů a robotů a udržovat informace o jejich míře poškození. Reaguje na zásah robota projektilem a na základě jeho typu provádí požadované akce, jako například zničení robota či jeho součástky.

Systém obstarává

- Při kolizi projektilu s robotem zaznamená udělené zranění
- V případě překročení určené hranice utrženého zranění zajistí reakci na základě typu robota
    - V případě hráčského robota notifikuje parts systém a dojde ke zničení součástky
    - V případě AI robota dojde ke zničení robota
- V případě, že hráč přijde o poslední součástku, zajistí zobrazení UI pro oživení hráče, a omezí jeho pohyb (dokud není opraven)
- Pokud oba hráči přijdou o všechny součástky, notifikuje **Game manager**, že jsou oba roboti zničeni a došlo k prohře

Realizace

HealthSystem je realizován jednou komponentou, nabízející funkci `Damage`, která umožňuje ostatním komponentám objekt zraňovat. Také přetěžuje funkci `OnCollisionEnter`, ve které reaguje na kolize projektilů s objektem, a na jejich základě upravuje stav životů trefeného objektu.

2.3. Managers

2.3.1. Level manager

Level manager zajišťuje načítání scén a přechod mezi nimi.

Realizace

Level manager je implementován formou Singleton návrhového vzoru, jako globálně přístupný samostatný GameObject, který se nachází v každé scéně. Slouží pouze jako rozhraní pro centralizované volání funkcí přechodu mezi scénami, a logikou s nimi spojenou (jako je například začernění obrazovky před přechodem mezi scénami).

2.3.2. Global game manager

Vzhledem ke způsobu, jakým zvolený engine pracuje se scénami, je nutné udržovat si některé globální hodnoty v objektu, který není smazán při přechodu mezi scénami. Úlohou **globálního game manageru** je tyto hodnoty udržovat napříč celou hrou.

Manager udržuje následující:

- Kolekci různých proměnných, které jsou uloženy ve slovníku.
- Stav inventáře hráčů, aby mohl být uchován při změně scény mezi jednotlivými navazujícími levely.
- Zvolené komponenty, ze kterých hráči postavili své roboty.
- Perzistenci, zajišťující uložení uživatelských dat, dosaženého progressu a zpřístupněných úrovní při vypnutí hry, včetně komunikace se Steam Cloud API, používanou na ukládání mezi zařízeními na distribuční platformě Steam
- Progression, udržující informace o tom, jaké úrovně hráči splnili a jaké komponenty jsou jim přístupné.

Realizace

GameObject globálního game manageru je vytvořen při spuštění hry, a skrze parametr DontDestroyOnLoad je jeho instance zachována mezi scénami. Implementuje návrhový vzor Singleton, a skrze něj k němu ostatní objekty přistupují.

Perzistence a ukládání

Perzistence bude zajištěna samostatným skriptem, který zajistí logiku ukládání, jenž bude spuštěn komponentou **GlobalGameManager** v případě, že uživatel chce svůj postup uložit.

Kromě přímé serializace veškerých relevantních dat na disk bude nutné zvolit dostatečný způsob enkrypce a obfuskace dat, aby nebylo jednoduché data falzifikovat a modifikovat. Pro potřeby projektu Silicomrades bude zvolen jednoduchý způsob XOR enkrypce na základě klíče obfuskovaného uvnitř projektu, který zabrání běžným uživatelům data relevantně modifikovat. Zastavit zkušené a dedikované uživatele v modifikaci dat je příliš náročné, a pro projekt to není nutné.
Skript zajistí také propojení se Steam Cloud API, která umožní ukládání dat uživatelů v cloudu distribuční platformy, na kterou bude v budoucnu projekt Silicomrades komerčně distribuován.

2.3.3. Game level manager

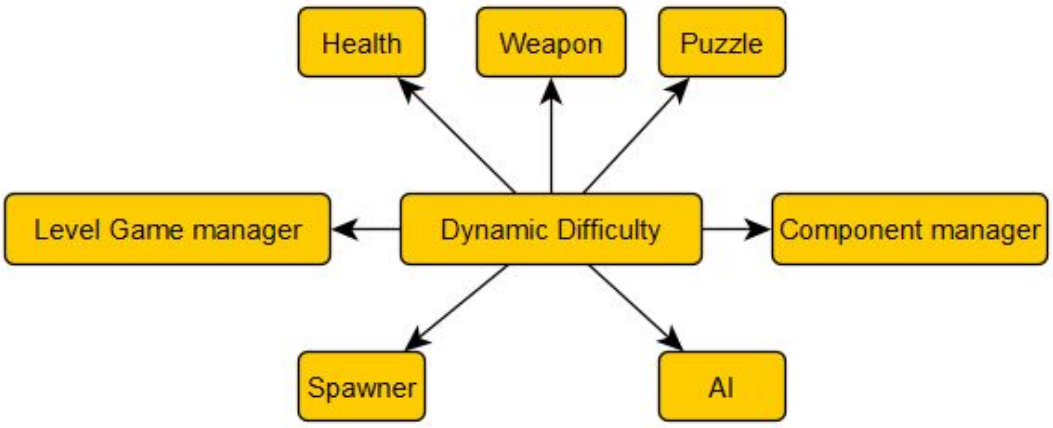
Tento manager zajišťuje spouštění skriptů, událostí a logiky vázané na jednotlivé úrovně. Detekuje splnění cíle a obstarává jednotlivé mechaniky specifické pro danou úroveň. Každá z úrovní má vlastní **Game level manager** a jejich kód se liší v závislosti na požadavcích dané úrovně.

Manager zajišťuje

- Splnění podmínek pro ukončení úrovně specifické pro danou úroveň:
    - Přežít daný čas
    - Posbírat určené objekty
    - Zničit určené objekty
    - Donést určené objekty na dané místo.
- Unikátní mechaniky dané úrovně. Mechaniky jsou následující:
    - Časový limit, po jehož uplynutí dojde k prohře v časem omezených úrovní
    - V některých úrovních hlídá úroveň hluku a pokud je překročena, dojde k prohře
- Spuštění událostí a skriptů specifických pro danou úroveň na základě akcí a pozice hráčů

Realizace

GameObject Game Level manageru je realizován komponentou **LevelXManager**, dědící ze třídy základního **Game manageru**, která implementuje funkcionalitu obecnou pro všechny úrovně (restart levelu, splnění zadaného úkolu, zastavení hry atd). Každá úroveň implementuje svojí verzi **Game Level manageru**, ve kterém je prováděna výše popsaná logika.



Obrázek zobrazuje, ze kterých všech systémů bude Dynamic Difficulty manager brát své vstupy.

2.3.4. Dynamic Difficulty Manager

Tento globální manager bude mít na starost vyhodnocování toho, jak se hráčům v průběhu aktuální úrovně daří a na základě získaných informací bude vypočítávat číselné ohodnocení, které bude sloužit jako vstup pro druhou úlohu tohoto manageru, kterou bude ovlivňování úrovně na základě vypočítaného skóre. Cílem je udržet hráče ve stavu, kdy pro ně hra je stále výzvou, ovšem ne tak velkou, aby se zdála býti nepřekonatelná. Tento přístup vychází z Flow principu (více např. zde https://www.gamasutra.com/view/feature/166972/cognitive_flow_the_psychology_of_.php?print=1).

Příklad vstupních dat:

- Uplynulý čas
- Aktuální stav životů jednotlivých robotů
- Jak moc hráči utrácejí součástky do volitelných systémů
- Kolik schovaných pickupů hráči našli
- Jak často si hráči navzájem součástky předávají
- Kolik střel hráčů minulo cíl

Možnosti následné úpravy světa:

- Změna toho, jak často padají náhradní součástky z poražených nepřátel
- Odemknutí či uzamknutí jednoduché / složité cesty
- Alternace počtu životů a způsobeného zranění nepřátel
- Změna počtu nepřátel či rovnou aktivace nebo deaktivace celých spawnerů
- Umělé prodloužení času nutného pro dokončení hrané mise

Dynamic difficulty manager tedy bude potřebovat komunikovat s následujícími systémy a objekty:

- GameLevelManager
- HealthSystem
- ComponentSystem
- PuzzleSystem
- Spawnery
- WeaponSystem

Realizace

Dynamic difficulty manager bude implementován formou **GameObjectu** přitomného ve všech scénách, stejným způsobem jako jednotlivé **LevelManagery**. Události ve hře bude detekovat za použití Observer návrhového vzoru, který bude provázaný celou aplikací. V první iteraci vývoje systému bude nejprve navržena a otestována naše vlastní heuristická funkce, hodnotící úspěšnost a dynamiku hry. Pokud se tento přístup ukáže jako neefektivní, či nebude poskytovat uspokojující výsledky, dojde k analýze, studiu a implementaci zjednodušené varianty jednoho z algoritmů navrhovaných v akademických textech týkajících se dynamické obtížnosti ve hrách, jako je například *Hamlet system (https://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-019.pdf),* či některý z dalších přístupů (https://www.hindawi.com/journals/ahci/2018/5681652/).

2.4. Music manager

Díky novému členu týmu, audio designérovi Janu Knotovi, je možné se nyní více zaměřit i na ozvučení a hudební doprovod hry Silicomrades. Přestože Unity engine sám o sobě nabízí systém pro přehrávání a mix audia, je tento v mnoha ohledech omezený. Rozhodli jsme se využít externí nástroj pro profesionální tvorbu dynamické muziky jménem FMOD, se kterým je náš nový audio designér zvyklý pracovat. FMOD sám nabízí knihovnu pro integraci do Unity, ovšem její poslední verze je relativně čerstvě vydaná a ukazuje se, že nedisponuje vším, co pro účely projektu potřebujeme. V takové chvíli zodpovědnost přebírá právě systém **Music manageru**.

Ten má na starost přehrávání doprovodné hudby v každé scéně a její bezchybné a plynulé přechody během načítání nových obrazovek. V ideálním případě tak muzika bude hrát od začátku do konce bez přerušení a zároveň bude reagovat na dění na obrazovce a přizpůsobovat se náladě právě hrané sekvence.

Podobně jako tomu je u **Dynamic difficulty manageru** tak bude třeba získávat informace z dalších systému, převážně z **Health systému** a **AI systému** v kombinaci s **Camera rigem**, díky které bude možné vyhodnotit informace o aktuální situaci na obrazovce a tomu přizpůsobit to, která ze zvukových stop vytvořených FMOD se aktuálně přehrává a při jaké hlasitosti. Zajímavých efektů bychom též rádi docílili pomocí dynamické změny výšek přehrávané stopy či modifikací toho, z jakého směru hudba zrovna hraje (například pokud

bude jen jeden z robotů kriticky poškozen, je možné modifikovat pouze pravý kanál stereo hudby).

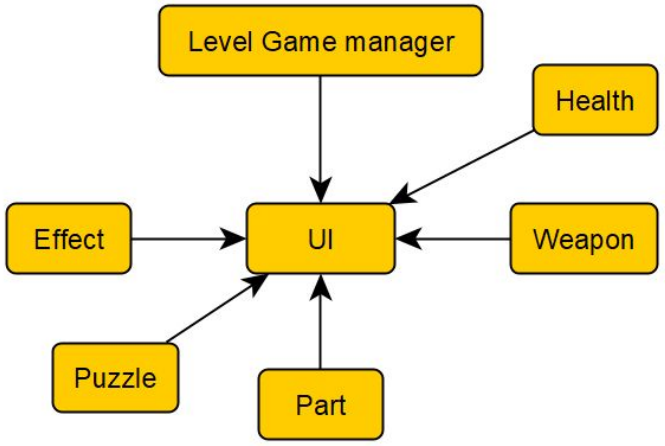
Realizace

Music manager bude realizován stejným způsobem jako **DynamicDifficultyManager** a bude úzce spolupracovat s API FMOD knihovny.

2.5. Animace

Pro animace postav používáme **Animator systém** implementovaný v použitém enginu Unity, není tedy nutné implementovat vlastní animační systém. **Animator systém** je obsluhován jednoduchým skriptem, který na základě rotace a stavu animovaného objektu zvolí a spustí správnou animaci.

Animace jsou tvořeny formou ručně kreslených a vybarvených sprite-sheets, které jsou posléze naskenované, digitálně vyčištěné a vložené do Unity **Animator systému**.



Obrázek zobrazuje, které všechny systémy potřebují mít výstupy či odezvu zobrazenou za pomoci uživatelského rozhraní.

2.6. Uživatelské rozhraní

Hlavní část uživatelského rozhraní je úzce spojena s parts systémem, neboť většina zobrazeného UI se týká inventáře, součástek a aktivních defektů. Během hry systém zajišťuje zobrazení stavu hráčova inventáře, obdržené defekty a další informace specifické pro jednotlivé úrovně. Také obstarává zobrazení a interakci s dalšími obrazovkami, jako je pause menu, obrazovka výběru komponent robota nebo obrazovka výběru úrovně a přechody mezi nimi.

2.6.1. Realizace

Uživatelské rozhraní je realizováno kombinací komponent a **GameObjectů**, využívajících rozsáhlého setu komponent **UnityUI**. Tento set komponent nabízí základní objekty používané pro interakci, zobrazení, zarovnání, škálování a řazení elementů uživatelského rozhraní.
Kromě nutnosti poskládat ve vizuálním UnityEditoru celé uživatelské rozhraní budou také implementovány sety komponent **UIController**, které reagují na události od herních komponent, a na jejich základě aktualizují zobrazené informace.

3. Příběh

Stěžejní částí projektu je epizodický formát a návrh příběhu hry. V této části bude přiblížen plánovaný průběh implementovaných epizod a jejich rozdělení do jednotlivých úrovní. Ke každé epizodě se pojí unikátní set implementovaných mechanik, které zde budou nastíněny. Dále zde také bude zmíněn celkový příběh a způsob, kterým je k jeho návrhu přistupováno.

3.0.1. Realizace

Každá z jednotlivých epizod sestává celkem ze sedmi scén. Tři z nich jsou vždy hratelné úrovně, ve kterých jsou přítomné výše popsané systémy a hráči s nimi interagují přes konfiguraci konkrétní scény (tj. umístění jednotlivých **GameObjectů**, jejich colliderů, rozmístění spawnerů a nepřátel atd.). Další tři jsou mezi-úrovňové obrazovky s briefingem a příběhem a sedmou scénou je vždy závěrečná obrazovka s obrázkem a vyústěním příběhu.

Předávání příběhu hráči je tedy realizováno formou intra při zapnutí hry a poté briefingovými obrazovkami s textem a tématickým obrázkem, který uvádí či posouvá zápletku dané epizody. V duchu environmentálního storytellingu bude příběh doprovázen i vhodnými grafickým assety.

3.1. Příběhové pozadí

3.1.1. Svět

Příběh se odehrává v převážně industriálním světě obývaným z velké části roboty. Ve státě, ve kterém se hra odehrává, vládne skoro až kafkovská byrokracie. Hlavními protagonisty hry jsou dvojice nízkorozpočtových levných robotů, kteří vycházejí z jedné staré Továrny, která byla proslulá produkcí nekvalitních výrobků.

3.1.2. Zápletka

Níže uvedená zápletka je pouze návrh a není v žádném případě definitivní.

Byl vydán zákon, který zakazuje staré a ekologicky neefektivní roboty. Naši protagonisté (a další roboti stejného typu) jsou tím pádem odvolání na sešrotování. Každý z nich reaguje jiným způsobem - někteří se vydají na úřad shánět výjimku ze zákona, jiní se rozhodnou

emigrovat do sousedního státu Retrolandie. Další se začnou schovávat, protestovat a vylepovat letáky.

Když se ukáže, že tento přístup není dlouhodobě udržitelný, vznikne Odboj starých neefektivních robotů s cílem sabotovat a destabilizovat vládu. Snaží se dokázat, že i vysocí pohlaváři jsou neefektivní, zastavit jejich komunikaci vyřazením potrubní pošty a zastavením těžby dřeva docílit nedostatku papírů na úřední formuláře.

I tento plán se ukáže jako nedostatečný a Odboj se rozhodne sehnat si armádu. Protože jsou ale neefektivní, a nikdy nebudou lépe vyzbrojení, spustí propagandistickou politickou kampaň, obsadí Továrnu a začnou zvedat počty neefektivních robotů s jedním cílem - přehlasovat momentální vládu a vyvolat referendum o zrušení zákona o neefektivních robotech. Referendum ale selže a dojde k revoluci, která končí převratem a nástupem neefektivních robotů k moci.

3.2. Epizody

Na konci softwarového projektu budou Silicomrades obsahovat tři epizody, které obsáhnou první část plánované zápletky. Každá epizoda je rozdělena do tří úrovní.

3.2.1. Epizoda I. - Útěk do Retrolandie

Tato epizoda byla úspěšně implementována v rámci prototypu na předmětu Vývoj počítačových her.

3.2.1.1. Příběh

Epizoda sleduje dvojici neefektivních robotů Carla a Jeremyho, kteří utíkají před zákonem o zákazu neefektivních robotů do sousedního státu Retrolandie, kde o ně bude postaráno.

3.2.1.2. Úrovně

Epizoda je rozdělena do tří úrovní, kde každá obsahuje unikátní herní mechaniku, která ji odlišuje od ostatních. Cíle jednotlivých úrovní, spolu s jejich unikátní mechanikou, jsou následující:

1. Dojít od začátku úrovně až ke hranicím a cestou zničit jeden ze dvou generátorů
2. Přežít na hranicích po dobu tří minut
3. Porazit hlavního úředníka v časovém limitu

3.2.2. Epizoda II. - Typická návštěva úřadu

Tato epizoda bude implementována jako součást práce provedené během absolvování softwarového projektu.

3.2.2.1. Příběh

Epizoda sleduje dvojici neefektivních robotů, kteří se rozhodli vypořádat s novým zákonem (a zákazem vlastní existence) tím, že se vydají na úřad zajistit si formulář udělující jim výjimku z tohoto zákona. Na úřadě zjistí, že se hodiny zastavily během pauzy na oběd

a žádný z byrokracií posedlých úředníků se nechce vrátit za přepážku, dokud podle hodin pauza neskončí. Roboti tedy vyrazí hodiny dát do pořádku a po jejich úspěšném opravení mohou oběhat několik přepážek a v klasicky byrokratickém pekle vyplnit nespočet formulářů. Formuláře poté nesou domů, přičemž se jich cestou snaží poztrácet co nejmenší množství.

3.2.2.2. Úrovně

Epizoda bude rozdělena do tří úrovní. Unikátní herní mechaniky zvolené pro každou z nich, které také určují cíl jednotlivých úrovní, budou následující:

1. Najít na jednom z náhodných míst na mapě stroj s hodinami a vrátit se zpět na start úrovně.
    a. Během toho nesmí dojít ke spuštění alarmu, který se aktivuje, pokud hodnota provedeného hluku roboty překročí určenou hranici.
    b. Provedený hluk se zvětšuje při střelbě robotů (hráčů i nepřátel).
    c. Hladinu hluku je možné vynulovat obětováním součástky na patřičném místě.
2. Získat určitý počet formulářů nalezením správných přepážek v daném pořadí před vypršením časového limitu.
3. Donést určitý počet formulářů do cílové oblasti před vypršením časového limitu.
    a. Hráči upustí jeden formulář na zem pokud jsou zasaženi.
    b. Upuštěné formuláře je možné znovu sebrat.

3.2.3. Epizoda III.

Třetí epizoda bude sledovat příběh další dvojice robotů, která se snaží vyřešit problém s nově vydaným zákonem. Jedním z prvních kroků po dokončení druhé epizody bude navrhnout příběh a mechaniky pro epizodu třetí. Tento krok je odložen až na pozdější část vývojového cyklu. Třetí epizoda by měla svou povahou a mechanikami doplňovat epizodu druhou.

4. Implementační požadavky

V této části následuje sumarizace specifikace minimálních požadavků pro splnění podmínek softwarového projektu. Seznam je rozdělený dle jednotlivých systémů a logických celků a popisován z pohledu hráčů. Funkce, které nejsou výsledkem přímé akce od hráče, jsou popsány z pohledu hry reagující na vzniklé situace.

Výčet funkčních požadavků je popisován vzhledem k prototypu. Funkce, které jsou implementovány v prototypu, ale vyžadují změnu z důvodu vylepšení hratelnosti, jsou zde také zmíněny. Funkce, které jsou již implementovány v prototypu a nevyžadují výraznější změny (například základní funkce **Camera rigu** či **Spawnerů**), jsou vynechány.

Vzhledem k iterační povaze vývoje her je možné, že během vývoje dojde ke změnám v konkrétním způsobu realizace plánovaných funkcí, či k jejich úplnému nahrazení. Celkový objem a náročnost plánovaných požadavků by ale měla být zachována. Ze stejného důvodu byly také některé z funkcí a požadavků navrhovaných v původním návrhu projektu nahrazeny jinou variantou, která se více hodila do celkové vize projektu.

4.1. [R01] Příběh

1. Hráčům je vhodně vysvětlený a uvedený koncept robotů ze stejné výrobní linky.
2. Hráči mají k dispozici na výběr z alespoň tří hratelných příběhů, které se liší hratelností, dějem a mechanikami.

4.2. [R02] Ovládání

1. Hráči mohou hru ovládat za použití dvou gamepadů, bez nutnosti použít klávesnici a myš.
2. Ovládací schéma se liší na základě robota, kterého daný hráč ovládá.
3. Ovládací schéma je dostatečně asymetrické a každý robot se chová trochu jinak.
    a. Jet robot se ovládá pomocí dvou joysticků, jedním pro pohyb a druhým pro míření.
    b. Crawl robot používá druhý joystick pro schopnost kotoulu a míří pouze směrem kterým se hýbe.
4. Hráči mohou hru pozastavit.
5. Hráči mohou za použití gamepadu ovládat veškeré uživatelské rozhraní.

4.3. [R03] Uživatelské Rozhraní

1. Uživatelské rozhraní je přehlednější a intuitivnější než v prototypu.
    a. Inventář je vždy zobrazen na obrazovce.
    b. Změna úkolu je zdůrazněna animací.
2. Obrazovka výběru komponent, na které hráči vybírají komponenty, ze kterých jsou postaveni jejich roboti.
3. Obrazovka výběru epizod.

4.4. [R04] Umělá Inteligence

1. Ve hře se nachází 4 nové druhy nepřátel, které výrazně mění hratelnost.
    a. Dron
    b. Heavy
    c. Shield
    d. Bomb
2. Umělá inteligence je schopná navigovat se po mapě bez větších problémů
    a. Pathfinding na základě ručně rozmístěných orientačních bodů.
    b. Steering behaviors zajišťující pohyb AI bez výrazně rušivých kolizí a zásek.

4.5. [R05] Komponenty a součástky

1. Hráči mohou použít součástky k interakci s prostředím
    a. Zpřístupnění alternativní cesty
    b. Aktivace pastí
    c. Aktivace turretů
    d. Souboj s vrchním úředníkem
    e. Vynulování úrovně hluku
    f. Prodloužení časového limitu na dokončení úrovně
2. Hráči mohou použít součástky k aktivaci speciálních schopností
3. Hráči si mohou vybírat z několika komponent při stavbě robota; funkční požadavky na jednotlivé komponenty jsou rozepsány v R08
    a. Jet robot
        i. Heavy Rounds
        ii. Fast Trigger
        iii. Knockback
        iv. Knoc-more-back
        v. More ammo
        vi. Aim assist
        vii. Spread stabilizer
        viii. Better engine
        ix. Active reload
        x. Laser sight
    b. Crawl robot
        i. Diagonal legs
        ii. Nimble unit
        iii. Quick reload
        iv. Roll
        v. Roll 2.0
        vi. Double barrel
        vii. Wider gun
        viii. Explosive ammo
        ix. Stronger amor
        x. Emergency shield

4.6. [R06] Obtížnost a zamezení snowballingu

1. Z některých zničených nepřátel padají součástky, které mohou hráči sebrat a tím si opravit komponentu
    a. Šance na vypadnutí součástky se zvyšuje podle toho, jak moc rozbití jsou hráči
2. Pokud je jeden z hráčů vyřazený, noví nepřátele přichází ve vlnách, aby z nich hráči měli možnost získat součástky na opravení

3. Po úrovních jsou rozmístěny součástky, které mohou hráči sebrat a tím se opravit
4. Je dynamicky počítáno aktuální hodnocení toho, jak si hráči vedou, a na jeho základě se mění počty a schopnosti nepřátel
5. Hra obsahuje interaktivní tutoriál, který umožní hráčům po částech pochopit a rovnou si i vyzkoušet jednotlivé mechaniky Silicomrades

4.7. [R07] Hudba

1. Hudba reaguje na změny ve hře
2. Zvuky neruší zážitek a sedí do celkové nálady hry

4.8. [R08] Pocit ze hry

1. Gamepady vibrují při významných událostech ve hře
    a. Větší výbuchy
    b. Obdržený defekt
2. Hráči mají asistenci při míření
3. Kamera se více posouvá směrem, kterým hráči míří
4. Na robotech se vizuálně projeví úroveň jejich poškození
5. Ve hře je možné dělat stabilní postup, který se ukládá a je k dispozici při dalším zapnutí

4.9. [R08] Efekty

1. Každá komponenta obsahuje unikátní bonus a defekt
    a. Heavy Rounds
        i. Větší zranění
        ii. Menší zranění
    b. Fast Trigger
        i. Větší kadence střelby
        ii. Střely mohou zranit hráče
    c. Knockback
        i. Schopnost tlakové vlny
        ii. Zbraň občas vystřelí sama od sebe
    d. Knoc-more-back
        i. Větší síla tlakové vlny
        ii. Tlaková vlna občas přitahuje místo odráží
    e. More ammo
        i. Více nábojů v zásobníku
        ii. Náhodný počet nábojů po přebití
    f. Aim assist
        i. Naváděné střely
        ii. Střelba zpomaluje
    g. Spread stabilizer

        i. Nižší rozptyl střelby
        ii. Obrovský rozptyl střelby
    h. Better engine
        i. Rychlejší pohyb
        ii. Nelze zastavit
    i. Active reload
        i. Možnost rychleji přebít
        ii. Nelze manuálně přebít
    j. Laser sight
        i. Laserový zaměřovač
        ii. Rozklepané míření
    k. Diagonal legs
        i. Možnost chodit diagonálně
        ii. Lze chodit pouze diagonálně
    l. Nimble unit
        i. Rychlejší pohyb
        ii. Pohyb se zrychluje a zpomaluje v náhodných intervalech
    m. Quick reload
        i. Rychlejší přebíjení
        ii. Pomalejší přebíjení
    n. Roll
        i. Schopnost kotoulu
        ii. Střelba má zpětný ráz
    o. Roll 2.0
        i. Delší kotoul
        ii. Ztráta imunity při kotoulu
    p. Double barrel
        i. Dva náboje v zásobníku
        ii. Menší zranění
    q. Wider gun
        i. Více broků při výstřelu
        ii. Méně broků při výstřelu
    r. Explosive ammo
        i. Zásahy odhazují nepřátele
        ii. Kratší dostřel
    s. Stronger amor
        i. Více životů
        ii. Robot je zraňován, když se nehýbe
    t. Emergency shield
        i. Delší imunita
        ii. Kratší imunita
2. Nepřátelé mohou způsobit krátké globální defekty, s omezenou dobou trvání
    a. Hráči se mohou navzájem zranit
    b. Hráči mají invertovaný pohyb

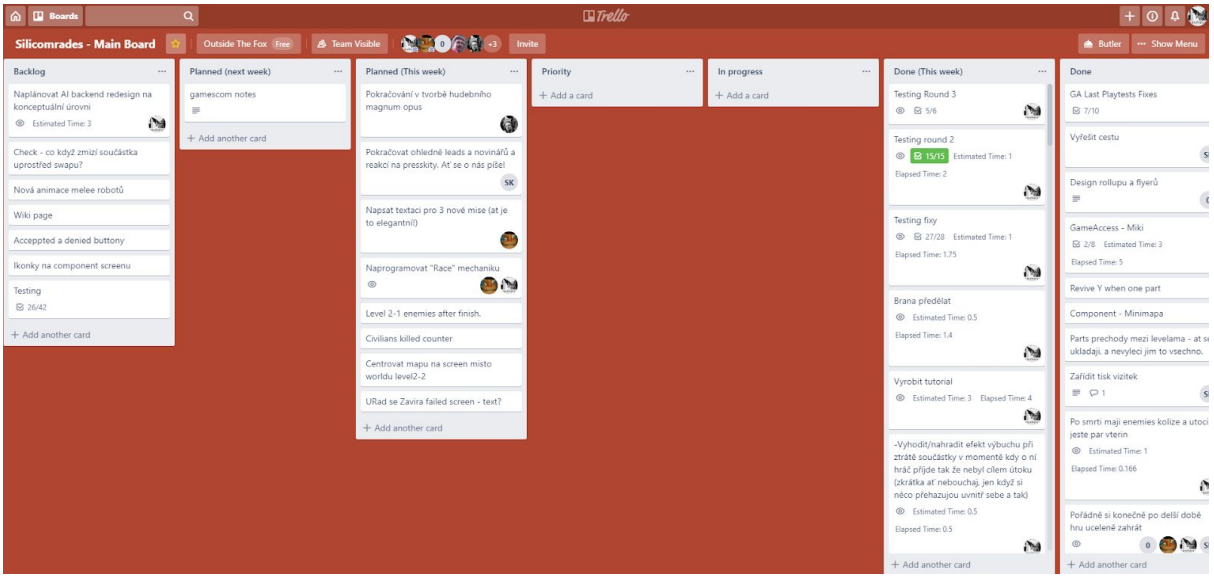
c. Obrazovka začne zrnit
d. Obrazovka se začne třást
e. Hráči se k sobě nesmí přiblížit
f. Hráči nemohou přestat střílet

5. Workflow

5.1. Organizace

Práce na projektu probíhá v týdenních iteracích. Každé pondělí se tým schází a synchronizuje videohovorem, během kterého každý odprezentuje úspěšně dokončenou práci a problémy, na jaké problémy narazil. Poté je rozdělena práce na další týden.
V případě, že některý ze členů stíhá svojí práci dříve, má možnost ujmout se některého z nepřidělených úkolů v backlogu.

Pro lepší organizaci rozdělených úkolů používáme nástroj Trello, virtuální nástěnku na kterou zaznamenáváme veškeré úkoly, které jsou naplánované, proběhly a nebo právě probíhají.



Snímek obrazovky nástěnky Trello, na které probíhá organizace práce týmu

5.2. Komunikace

Pro potřeby běžné komunikace používáme skupinový chat Messenger, ve kterém řešíme akutní problémy které nastaly. Pro videohovory a synchronizaci je používána platforma Skype.

5.3. Contingency plan

Tým Outside the Fox spolu v aktuálním složení již pár měsíců funguje a nic nenasvědčuje tomu, že by snad některý z členů měl v plánu se na projektu přestat podílet. Přesto zde uvádíme pro úplnost contingency plán, aby bylo jasné, jak by tým v případě ztráty některého z členů postupovat. Protože jsou povinnosti každého člena týmu jiné, je ke každé potenciální ztrátě třeba přistupovat individuálně. V případě, že tým opustí:

- Jan Holan
    - Vedení projektu převezme Mikuláš Hrdlička, stejně tak pozici hlavního designéra. Pokračovat bude podle plánů sepsaných zde ve specifikaci. Je u projektu od začátku a díky svým znalostem programování má dobrý přehled o tom, co všechno vývoj jednotlivých plánovaných features obnáší.
- Mikuláš Hrdlička
    - Programování naplno převezme Jan Holan. Tím pádem bude nutné slevit z plánované časové alokace na design dalších mechanik, každopádně softwarový projekt bude stále realizován ve slíbeném rozsahu.
- Jakub Eremiáš
    - V případě ztráty hlavního grafika nebude snaha o nalezení jeho náhrady - styl jiného umělce by pravděpodobně bylo velmi těžké zkombinovat s již vytvořenou vizuální identitou Silicomrades. Pro dokončení projektu by tak byly zrecyklovány již existující assety, kterých prozatím Jakub Eremiáš stihnul vyrobit nemalé množství.
- Štěpánka Kološová
    - V případě, že by tým opustila Štěpánka Kološová, tak by design uživatelského rozhraní přebral pravděpodobně Mikuláš Hrdlička či Jakub Eremiáš. Jakub by byl na danou práci vhodnější, každopádně pozice hlavního grafika je velmi časově náročná a je otázkou, zda-li by stihnul ještě nějaké povinnosti navíc.
- Jan Knot
    - Pokud tým přijde o svého zvukového designéra, dostane se vlastně zpět do složení, v jakém byl vývoj Silicomrades před rokem zahájen. Zvuky a muzika by se tím pádem stali povinností všech zbylých členů týmu a nejednalo by se o autorské assety nýbrž stopy s vhodnou licencí posbírané napříč internetem.

5.4. Vedlejší aktivity

Krom práce na výše zmíněných herních prvcích a mechanikách tento projekt obnáší i velké množství testování s nestranným publikem, administrace, managementu, marketingu, logistiky a sběr a následnou analýzu zpětné vazby. S projektem plánujeme na konci srpna navštívit Gamescom, jehož výsledek stanoví směr, kam se s hrou dále vydáme. I v případě neúspěchu budeme na projektu samozřejmě pokračovat, dokud nedosáhne požadovaného rozsahu.

5.4.1. Uživatelské testování

Uživatelské testování je důležitou součástí vývoje každé hry. Umožňuje tvůrcům poskytnout přímý vhled do přemýšlení hráčů a ukazuje na nedostatky, které jsou autory hry často přehlíženy. Uživatelské testování v rámci vývoje hry Silicomrades bude probíhat v několika seancích, vždy bude doplňovat konkrétní Milestone (viz níže), bude probíhat ale také za každou větší změnou, posunem ve vývoji a samozřejmě také v situacích, kdy se v týmu na konkrétním prvku neshodneme a necháme o vhodnosti daného řešení rozhodnout samotné hráče.

Průběh testování

Průběh našeho testování vychází ze standardních metodik uživatelského testování. Jelikož se jedná o hru pro dva hráče, je potřeba pracovat minimálně se třemi zúčastněnými, přičemž dva z nich jsou testovaní a jeden testující. V ideálním případě se zde nachází také pozorovatel, který si všímá reakcí testovaných na prováděné úkoly, na jejich pochopení a zaměřuje se také na non-verbální komunikaci. Samotné testování se může lišit podle toho, zda-li již testovaní mají předchozí zkušenosti s touto hrou či nikoliv, nicméně jeho průběh je vždy stejný. Testující zadá testovaným úkol (např. dokončit level), následně testovaní úkol plní a testující zodpovídá případné dotazy. Po té následuje polostrukturovaný rozhovor, který vede testovaný s testujícími ohledně zkoumaného problému. Rozhovor je polostrukturovaný, protože je každá jednotlivá testovací seance organizovaná s cílem získání zpětné vazby na konkrétní prvek, proto je potřeba ke správnému vyhodnocení, aby všechny testované páry zodpověděly předem připravené otázky, nicméně je zde prostor i pro doplňující otázky či dovysvětlení.

Zájemce o testování hry je možné oslovit skrze sociální sítě, newsletterem odeslaným na lidi, kteří nám zanechali pro takovéto případy kontakt na konferencích, či lze oslovit studenty Vývoje her. Při jednom testování chceme pracovat minimálně s deseti dvojicemi, kdy má každá dvojice 30 minut na vyřešení testovaného problému a 30 minut na polostrukturovaný rozhovor.

5.4.3. Testování užitečnosti

V uživatelském testování budeme kombinovat metody A/B testování s testováním užitečnosti (usefullness testing), které využijeme především při testování uživatelského rozhraní. Testování užitečnosti totiž umožní hlubší vhled do toho, jak hráč uživatelské rozhraní vnímá vzhledem ke splnění daného úkolu ve hře a díky hloubkovému rozhovoru tak můžeme například zjistit, že je námi vytvořené uživatelské prostředí sice pochopitelné, nicméně by hráč ocenil jiný přístup, který by pro něj byl užitečnější.

5.4.2. A/B testování

Metoda A/B testování nám umožní porovnat zpětnou vazbu dvou testovaných skupin, které testují dvě jiná řešení jednoho problému. Ačkoliv je naše uživatelské testování spíše kvalitativního rázu, zde můžeme použít prvky kvantitativních metod a v případě dostatečného množství dat použijeme variantu, která se mezi testovanými osvědčila lépe.

V neposlední řadě je zde potřeba zmínit expertní testování, tedy získávání zpětné vazby od vedoucího projektu. Ta je velmi důležitá především proto, že na rozdíl od testovaných, nám umí expert sdělit nejen, že je něco špatně, ale také zdůvodnit proč.

5.5. Roadmap

5.5.1. Milestone 1

Prvním milníkem je dokončení druhé epizody a cesta na Gamescom. Kvůli konferenci bude nutné se zaměřit kromě samotného vývoje i na polishing, prezentaci, domlouvání meetingů s potencionálními investory, marketingem a konsolidací celkové vytvořené hry do produktu, který působí dokončeně a je možné jej na konferenci prezentovat. Z důvodu nedostatku času je možné, že kvůli konferenci bude nutné implementovat některé ze systémů způsobem, který je funkční pro požadované případy použití, ale není dostatečně obecný či elegantně implementovaný a bude nutné jej v dalších částech projektu refaktorovat.

Milestone 1 Aftermath

V době obhajoby specifikace byl už první milník projektu zdárně dokončen a k nově implementovaným funkcím byl získán cenný feedback od důležitých lidí herního průmyslu. Zpětná vazba týkající se „umělecké“ stránky hry zde nebude uvedena. Každopádně nejčastěji zmiňovanými problémy se zdají býti následující:

- Samotný pocit z hraní potřebuje být více „juicy“, což je termín vyjadřující pocit satisfakce, kterou hráč ze svých akcí provedených ve hře má (podrobněji třeba viz. zde https://www.youtube.com/watch?v=Fy0aCDmgnxg). Jedná se tím pádem převážně o designovou výzvu, protože „juiciness“ často záleží na malých detailech, jaké často hráč ani vědomě nevnímá, proto se obtížně testuje.
- Prezentované verzi hry chybí nějaký pocit dosaženého postupu, odemykání nových úrovní či komponent. Tento nedostatek je prozatím způsoben nedostatkem obsahu a systém umožňující ukládání a načítání dosažených unlocků bude v dalším milníku implementován.
- Je velmi náročné se do hry poprvé dostat a jednotlivé principy Silicomrades pochopit v plném rozsahu. Tento problém budeme v dalším milníku adresovat pomocí vytvoření interaktivního tutoriálu (narozdíl od statického textu doplněného gify, jakým jsou principy vysvětleny nyní) a pokud se ten po testování ukáže jako nedostatečný, tak pomocí dynamického systému nápovědy, který se bude snažit detekovat chvíle, kdy si hráči nejsou jistí, co se děje, a zobrazí jim vhodnou radu.
- Pro ozvláštnění hry by mohlo býti zajímavé poskytnout hráčům různé časově vylepšené powerupy, jaké by bylo možné získat v průběhu hektických bojů. Tento názor se objevoval ve zpětné vazbě spíše sporadicky, nicméně za předpokladu, že se nám v druhém milníku podaří své záměry splnit s předstihem, pustíme se i do designu a implementace systému takovýchto dočasných vylepšení.

Milestone 1 shrnutí práce

- Návrh a implementace **Component systému**, včetně **Effect systému**
- Využití externí knihovny pro možnost ovládání různými periferiemi a zapracování do **Input manageru**
- Návrh, design a implementace druhého příběhu hry, sestávajícího ze 3 misí
- Přizpůsobení UI pro ovládání za pomoci gamepadů
- Přepracování UI pro větší intuitivnost a otestování výsledků

Milestone Timeline

- 1. měsíc
    - **Component systém** a **Input manager**
- 2. měsíc
    - UI
    - Druhý příběh hry
    - Testování

5.5.2. Milestone 2

Druhý milník následuje po zpracování zpětné vazby z konference, obepsání potencionálních investorů a followupu dle dosažených výsledků. Také dojde k návrhu a dokončení třetí epizody, na základě mechanik a odezvy druhé epizody. Dalšími plánovanými cíli jsou i vylepšení chování AI a implementace dynamické obtížnosti. Všechny změny budou podrobeny důkladnému testování na nejméně dvou testovacích sezeních, která budou probíhat podle principů uvedených v příslušné kapitole.

Milestone 2 shrnutí práce

- Kompletní přepracování starého **AI systému**, včetně pathfindingu a steering behaviors
- Návrh a implementace **Dynamic difficulty systému**
- Návrh a implementace **Progress systému**, který se bude starat o ukládání stavu hry na disk a umožní hráčům strukturovaný postup hrou
- Návrh, design a implementace interaktivního tutoriálu, který hráčům vysvětlí přehledně herní mechaniky
    - Pokud se při testech ukáže tento krok jako nedostatečný, vznikne **Dynamic help systém**, který se bude snažit při hře detekovat stavy, kdy si hráči neví rady, a bude při nich zobrazovat vhodnou nápovědu
- Testování
- Umělecká část projektu (tvorba prostředí, postav a jejich animace. Tvorba muziky a zvukových efektů)
- Business část projektu (dohody s potenciálními investory, marketing…)

Milestone Timeline

- 1. měsíc
    - **AI systém** a **Progress systém**
- 2. měsíc
    - Interaktivní tutoriál (potažmo **Dynamic help systém**)
    - **Dynamic difficulty systém**
- 3. měsíc
    - Buffer (pro případ, že by předchozí dva měsíce nešly podle plánu)
    - Testování

5.5.3. Milestone 3

Cesta ke třetímu milníku je naplněna převážně zpracováním dokumentace, polishingu, dolaďováním hry a dalším uživatelským testováním, díky kterému bude získána závěrečný zpětná vazba, která bude zpracována a implementována do hry. V této fázi již bude projekt v beta verzi a místo dalšího přidávání systému a funkcí bude hra pouze plněna obsahem.

Milestone 3 shrnutí práce

- Návrh, design a implementace třetího příběhu hry, sestávajícího ze 3 misí
- Návrh a implementace **Music manageru**, který se stará o hudbu reagující na dění na obrazovce a tvorba příslušných hudebních stop
- Pokud zbyde čas navíc, dojde na návrh a implementaci **Powerup systému**, který se na základě feedbacku z prvního milníku ukázal z pohledu hráčů jako velmi zajímavý
- Refaktoring a polishování zážitku
- Návrh a implementace kroků přispívající k „juiciness“ hry na základě výstupů z testování v předchozím milníku
- Testování
- Tvorba dokumentace
- Umělecká část projektu (tvorba prostředí, postav a jejich animace. Tvorba muziky a zvukových efektů)

Milestone Timeline

- 1. měsíc
    - Tvorba třetího příběhu a podpůrných systémů
    - **Music manager**
    - „juiciness“ a testování
- 2. měsíc
    - **Powerup systém**
    - Dokumentace
    - Testování

5.6. Rozdělení práce

Tým aktuálně sestává z 5 členů na těchto pozicích

- Jan Holan <jan.holan96@gmail.com>

    - MFF UK - Magisterské studium Počítačová grafika a vývoj počítačových her
    - Team Lead, Lead Designer, Programmer
    - Jeho hlavní náplní práce je vést tým, rozdělovat práci, a komunikovat s potencionálními investory a médii. Rozhoduje o čemkoliv, co se týká herního designu hry, a pomáhá se samotnou implementací.
- Mikuláš Hrdlička <thehrdlicka@gmail.com>
    - MFF UK - Magisterské studium Počítačová grafika a vývoj počítačových her
    - Lead Programmer, Designer, QA
    - Jeho hlavní náplní práce je návrh a implementace většiny architektury a backendu. Rozhoduje o věcech týkajících se samotné implementace a kódu. Pomáhá také s designem a testováním aplikace.
- Štěpánka Kološová <stepankakolosova@gmail.com>
    - FF UK - Magisterské studium Studia nových médií
    - Artist, Level-designer, Marketing, Management
    - Její hlavní náplní práce je tvorba vektorové grafiky pro uživatelské rozhraní a propagační materiály. Obsluhuje sociální sítě a rozhoduje o věcech spojených s marketingem. Také zodpovídá za uživatelské rozhraní, a má poslední slovo ve věcech, které se ho týkají.
- Jakub Eremiáš <j.eremias@seznam.cz>
    - Fakulta designu a umění Ladislava Sutnara ZČU
    - Lead Artist, Animator
    - Jeho náplní práce je tvorba animací, objektů a artu. Zodpovídá za vizuální stránku hry, a má poslední slovo ve věcech, které se jí týkají.
- Jan Knot <knotjan.audio@gmail.com>
    - FAMU
    - Sound Designer
    - Jeho náplní práce je tvorba veškeré hudební stránky hry. Zodpovídá za a rozhoduje o zvukových efektech, hudbě a způsobu, jakým se dynamicky mění a projevuje.

Práce těch členů týmu, kteří se nepodílí na programovací části aplikace, je samozřejmě svým rozsahem úměrná tomu, co je třeba naprogramovat. Vzhledem k účelu specifikace a cílům softwarového projektu zde ovšem umělecké části vývoje (grafika a audio) nejsou rozepsány do detailů. V jednoduchosti se dá ovšem říci, že skoro každý nový naprogramovaný systém je doprovázen nějakou grafikou či vhodně zvoleným zvukem, stejně tak nově implementované mechaniky velmi často pro lepší orientaci hráčů vyžadují nějakou vizuální či audio odezvu.

6. Příloha 1. - Unity Game Engine

V této kapitole je zběžně představen použitý herní engine Unity, jeho funkce a způsob, kterým se s enginem pracuje, stejně jako hlavní používaná terminologie. Nejedná se o detailní a přesný popis, ale o stručné přiblížení námi použité technologie, které je relevantní pro tuto specifikaci a náš projekt.

Herní engine Unity zprostředkovává základní nástroje, základní architekturu a komponenty potřebné pro vývoj a distribuci herního software. Zajišťuje low-level funkcionalitu, která je společná pro většinu herního software, a její zprostředkování nezávisle na platformě pro kterou je hra vyvíjena, aby byl umožněn vývoj a jednoduché nasazení i mezi platformami.

Funkcionalita zprostředkovaná Unity, která je relevantní a využívaná projektem Silicomrades, je následující:

- Editor umožňující vkládání **GameObjektů** a jejich komponent do scén.
- Zajišťuje hlavní herní smyčku, která je spolu s **Entity-Component** návrhovým vzorem hlavním stavebním kamenem herního vývoje.
- Obsluha a rendering scén a herních objektů na ní se nacházejících. Jejich vykreslování na základě pozice objektů ve scéně, a pozice kamery.
- 2D fyzika a kolize objektů, na základě tvaru jejich colliderů, pozice a směru pohybu. Zprostředkuje eventy typu `OnCollisionEnter`, na které objekty ve hře reagují, či spouští skripty dle potřeb hry.
- UI - rendering, pozice, scaling na základě rozlišení a zprostředkování eventů pro interakci uživatelů s rozhraním.
- Animační systém, který za použití FSM zajišťuje přechody a správné přehrávání animací jednotlivých objektů.
- Resources a content management - nahrávání obrázků, modelů, hudby a dalších assetů, práce s nimi v editoru, i jejich použití v komponentech.
- Základní komponenty, umožňující GameObjektům využívat výše zmíněné funkce. Příkladem jsou například tyto základní komponenty, používané téměř u všech objektů v projektu:
    - **Transform** - umožňuje objektu vyskytovat se na daných souřadnicích ve scéně.
    - **Rigidbody2D** - zajišťuje 2D fyziku a detekuje kolize s ostatními objekty.
    - **Collider2D** - určuje rozměry objektu pro kolize.

Mimo tyto zmíněné systémy a funkcionalitu poskytuje Unity i mnoho dalších, od 3D grafiky, přes osvětlení až po online multiplayer či VFX, které ale naším projektem nejsou využívány.

6.1. Architektura

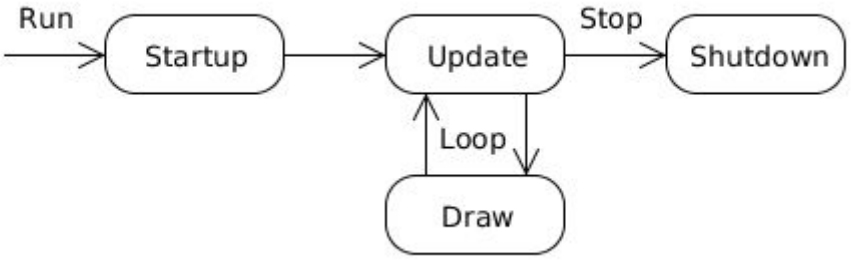
Herní engine Unity a naše práce s ním, je postavena na dvou základních návrhových vzorech velmi využívaných v herním vývoji - vzory Entity-Component a Game loop.

6.1.1. Entity - Component

Každý objekt, který se ve hře nachází, je základní instanci třídy **GameObject**. Veškerá další funkcionalita je objektům přidávána za použití různých komponent. Hlavní část vývoje, která bude na projektu prováděna, je tvorba skriptů realizujících komponenty, které dodávají objektům funkcionalitu potřebnou pro námi navržené mechaniky a systémy.

6.1.2. Game loop

Návrhový vzor používaný v naprosté většině herního software. Game loop zajišťuje časově nezávislý průběh hry a vykreslování objektů. Cyklus je zobrazený na obrázku níže.



Základní implementace game loop. Unity používá detailnější game loop s více procedurami, high-level idea ale zůstává stejná.

Hlavní myšlenkou je volání metod `Update` a `Draw` pro každý **GameObject** v cyklu a na základě uplynulého času od posledního cyklu a vstupů od uživatele měnit pozice, počítat fyziku, animovat, reagovat na vzniklé kolize a celkově zařizovat veškerou logiku, kterou je nutné spočítat pro průběh hry.
V `Draw` cyklu poté dochází k překreslení celé obrazovky, na základě nových pozic a rotací a animací objektů z posledního `Update` cyklu.
`Draw` metodu zajišťuje Unity a není nutné do ní zasahovat. Jediná forma, kterou bude do `Draw` metody během projektu potencionálně zasahováno, bude formou psaní shaderů, pokud to bude vizuální stránka hry vyžadovat.

6.2. Komunikace komponent a GameObjectů

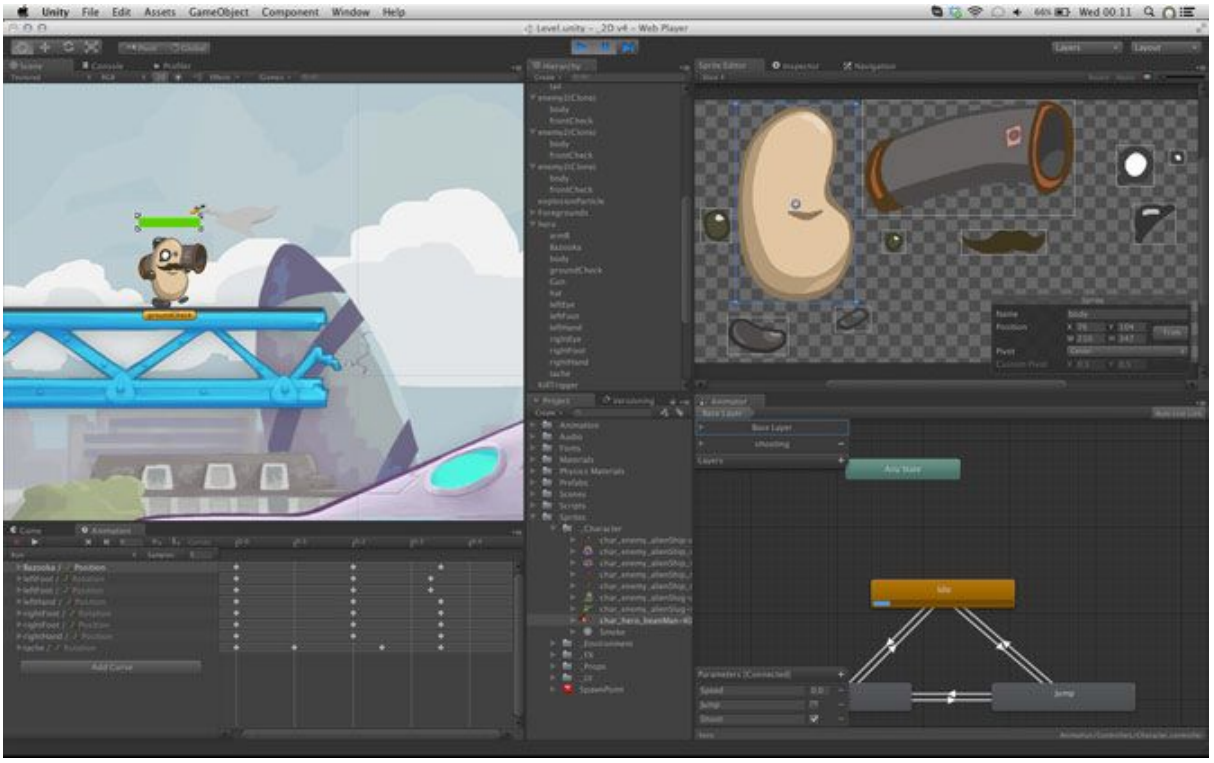
Unity nabízí několik možností způsobu komunikace komponent mezi sebou. Při vývoji projektu Silicomrades jsme se rozhodli používat ve většině případů přímé reference na třídy komponent, které je možné ručně nalinkovat skrze Unity Editor, nebo získat za použití metody `GetComponent<Type>` na **GameObject**, jenž komponentu vlastní.
Dalším podporovaným způsobem komunikace, převážně se základními komponentami Unity (jako je například **Rigidbody2D**), je overriding metod ze základní třídy **MonoBehavior**. Tato třída nabízí metody typu `OnTriggerEnter`, `OnCollisionEnter`, `OnMouseHover` a podobně. Vstupem této metody je reference na **GameObject,** kterého se událost týká, a tudíž lze jednoduše na jeho základě ustanovit komunikaci mezi relevantními objekty.

Další možností, využívanou převážně v Observer návrhovém vzoru, který bude v rámci projektu také implementován, je použití delegátů, nativních pro používaný jazyk C#. Unity nabízí také rozhraní **UnityEvents**, které se podobá delegátům a obsahuje integraci do Unity Editoru pro snadnější propojení objektů při tvorbě scén.

6.3. Workflow

Hlavní náplní provedené vývojářské a programovací práce na projektu je tvorba nových komponent pro **GameObjecty**, které jim dodávají funkcionalitu požadovanou naší hrou. Komponenty jsou tvořeny formou psaní skriptů v jazyce C#, děděním třídy **MonoBehavior**, a implementací logiky převážně probíhající v metodě `Update`, která obstarává chování objektů každý frame. Pro hráčovu postavu to může například znamenat reagovat na event o kolizi od komponenty **2DCollider**, dotazovat se **InputManageru** zda nedošlo k nějakému vstupu od hráče a nebo vytvořit nový **GameObject** střely na dané pozici a udat jí správný směr.
Různé managery jsou realizovány také formou **GameObjectů**, které ale neobsahují žádnou komponentu **Renderer** ani **Rigidbody**, nejsou tedy zobrazeny ve světě. Obsahují ale komponenty **MonoBehavior** (námi napsaných skriptů), které reagují na eventy, či mají vlastní metodu `Update`, která každý frame ovlivňuje hru.

Další důležitou náplní naší práce s enginem je tvorba assetů, scén, levelů a uživatelského rozhraní v Unity Editoru zobrazeném níže.



Unity Editor rozhraní

Velká část vývojového času je věnována úpravě a hledání vhodných konstant a způsobů realizace (a návrhu) herních mechanik. Výše popsané jsou pouze čistě vývojové činnosti relevantní pro práci s Unity, pro bližší představu čtenáře o pracnosti projektu.

6.4. Komponenty a Moduly Unity

Následující seznam obsahuje výčet komponent a modulů, které jsou zmiňované v textu specifikace a jsou již implementované jako základní komponenty či funkce Unity. Tyto komponenty nejsou implementovány jako součást práce na projektu Silicomrades, nýbrž je s nimi pracováno a jsou používány skrze celou aplikaci.

- Moduly
    - **Raycast** - Zajišťuje raycasting, vhodný například pro detekci viditelnosti mezi objekty.
- Komponenty
    - **Transform** - Komponenta zajišťující pozici a rotaci objektu ve scéně.
    - **Rigidbody2D** - Zajišťuje fyziku, pohyb a kolize 2D objektů ve scéně mezi sebou. Pro zadanou rychlost a směr simuluje pohyb objektu a spolu s komponentou **Collider2D** reaguje na kolize.
    - **Camera -** Komponenta starající se o zobrazení scény uživateli.
    - **Collider2D** - Nabízí collidery pro detekci kolizí mezi objekty. Objekty mohou mít collidery neprůchodné, nabízející eventy `OnCollision`, nebo collidery typu trigger, které nabízí eventy `OnTrigger`.
    - **SpriteRenderer** - zajišťuje vykreslení a obarvení obrázků a spritů.
    - **Animator** - poskytuje FSM, rozhraní a editor pro přechody, přehrávání a tvorbu animací v Unity. Za jeho pomoci je možné vytvořit animační klipy, které ovlivňují hodnoty na kterémkoliv z jiných komponentů v závislosti na čase. (Například změnu spritu ve **SpriteRendereru**, nebo posunutí **GameObjectu** o nějaký offset)
    - **TextMeshPro** - Komponenta zobrazující text. Umožňuje nastavení velkého množství parametrů, včetně podpory rich text markup language.
    - **UnityUI** - set komponent, které zajišťují zakládní elementy a práci s uživatelským rozhraním. V projektu jsou používané následující komponenty:
        - **UIScaler** - zajišťuje přeškálování rozhraní na základě referenčního rozlišení ve kterém bylo vytvořeno, a rozlišení, ve kterém je hra spuštěna.
        - **Canvas**, **Panel** - Komponenty fungující jako kontejner pro další elementy.
        - **Button** - Nabízí UnityEvent callbacky spouštěné při detekovaném kliknutí na element.
        - **Image** - Zobrazuje obrázek.
        - **Grid Layout** - Automaticky seskupí a zobrazí přiřazené elementy do mřižky, dle zadaných parametrů rozestupů, velikostí atd.