Running with modifiers  
Zpracování prototypu hry

# Přehled klíčových systémů

Za chod hry zodpovídá několik herních systémů. Tyto systémy fungují nezávisle na sobě a komunikuje se s nimi pomocí eventů, aby bylo možné změnit průběh hry přímo v enginu, bez nutnosti dělat změny v kódu.

### Game Manager

Držitel všech ostatních systémů. Pouze zobrazuje FPS.

### Level Manager

Systém, který hlídá stav a zobrazování výsledků hry. Obsahuje pod-systém **Level Builder**, který vytváří úroveň podle scriptable objectu **Level Set. Level Builder** vytvoří instanci prefabu **Platform** pro každou sekci úrovně v **Level Setu**.

### Level Styler

Zajišťuje změnu vzhledu hry pomocí scriptable objectu **Level Scheme**, ve kterém lze nastavit barvy různých objektů.

### Input Manager

Čte uživatelský vstup. Vyvolává události při začátku, změně a konci vstupu.

### Monetization Manager

Stará se o načítání a zobrazování reklam.

### Audio Manager

Zajišťuje přehrávání zvukových klipů ze scriptable objectů **Sound**. **Audio Manager** reaguje na událost vyvolanou na **AudioRequestEventChannelu**.

# Přehled klíčových komponentů

### Unit

Komponent jednotky, který má na starost životní cyklus jednotky a vyhledávání cílů při konfrontaci s nepřátelskými jednotkami. Komponent vyvolává události téměř při každé akci, aby bylo možné rozšířit funkcionalitu přímo v enginu.

### UnitGroup

Komponent skupiny jednotek. Tento komponent řeší matematické operace a vytváření nových jednotek. Komponent vyvolává události při změně počtu jednotek a když je skupina prázdná.

### Gate

Komponent brány. Řeší zobrazení matematické operace hráči. Komponent vyvolává událost při překročení hráčovou jednotkou, která předává matematickou operaci brány hráčově skupině jednotek k provedení.

### Platform

Komponent sekce úrovně, který nastavuje matematické operace svých bran a vytváří skupinu nepřátelských jednotek. Komponent vyvolává události téměř při každé akci.

### EventChannelListener

Komponent, který umožňuje v enginu nastavit obecnou reakci na vyvolání EventChannelu.

### AsyncEventSequence

Komponent, který umožňuje v enginu nastavit obecnou sekvenci událostí, která se spustí „asynchronně“ použitím Coroutine.

# Přehled klíčových skriptů

### Level Set

Scriptable object, který uchovává informace o úrovni a její sekce. Každá sekce obsahuje dva matematické výrazy a počet nepřátelských jednotek v dané sekci.

### Level Scheme

Scriptable object, který uchovává barevné schéma hry. Lze nastavit barvu pro pozadí, zem, brány, hráčovy jednotky a nepřátelské jednotky.

### EventChannel<T>

Scriptable object, který zajišťuje komunikaci mezi objekty, které na sebe nemohou získat odkaz.

Jelikož nelze vytvářet scriptable objecty, které mají obecnou třídu, tak je tato třída abstraktní a pro každý datový typ, který je potřeba, se vytvoří vlastní *balící* třída.

**AudioRequestEventChannel** je třída, která dědí **EventChannel<Sound>**.

### GameObjectPool

Scriptable object, který slouží ke sdílení jednoho object poolu mezi různými komponenty, aby nemuselo existovovat několik různých object poolů, které obsahují *stejný* objekt.

### Reference<T>

Obecná třída, která se stará o získání odkazu na objekt, na který nelze nastavit odkaz ručně v enginu – například když nějaký prefab potřebuje odkaz na objekt ve scéně.

# Prefaby

### Character

Model postavy. Je vykreslován s černým obrysem, aby jednotky nesplývaly s prostředím. Jde pouze o grafický prvek, nic nedělá.

### Player Unit

Objekt jednotky hráče. O životní cyklus a řízení instance se stará komponent **Unit**. Instance jsou brány z object poolu Player Pool.

### Enemy Unit

Objekt nepřátelské jednotky. Variace **Player Unit** – liší se pouze v nastavených událostech na komponentu **Unit**. Instance jsou brány z object poolu Enemy Pool.

### Gate

Objekt brány. Tvořena grafickým objektem, canvasem s textem, který zobrazuje matematickou operaci, a colliderem, který detekuje překročení.

### Platform

Objekt sekce úrovně. Kontejner pro instance prefabu **Gate** a nepřátelské jednotky. Jedná se o hlavní stavební blok úrovně. Instance jsou brány z object poolu Platform Pool.

# Postup

Začal jsem výstavbou úrovní. Vytvářet každou úroveň ručně by bylo časově náročné, proto jsem vytvořil systém, který úrovně sestaví podle dat ve scriptable objectu **Level Set**. Level designér by tímto způsobem mohl jednoduše vytvářet úrovně pouze nastavováním matematických operací a počtu nepřátelských jednotek v libovolném počtu sekcí.

Dále bylo třeba vyřešit vytváření jednotek. Jelikož se jednotky vytváří ve velkém množství, bylo potřeba využít Object Pooling, aby se zabránilo mikro lagům.

Aby mohlo existovat několik spawnerů nezávislých na sobě, object pooly jsou uchovávány ve scriptable objectu **GameObjectPool**, aby mohly být sdíleny mezi spawnery. Každá jednotka patří do skupiny jednotek **UnitGroup**, do které je automaticky přidána po vytvoření.

Následoval pohyb jednotek. Aby hra podporovala nekonečný herní mód, byl pohyb jednotek vyřešen tak, aby hráčovy jednotky stály na místě a pohybovaly se jen vlevo a vpravo. Všechny ostatní objekty ve scéně se naopak pohybují směrem k hráčovým jednotkám, což z pohledu hráče vytváří iluzi, že se hráč pohybuje dopředu. Kvůli tohoto přístupu bylo třeba ošetřit pár okrajových případů, kdy tato iluze byla porušena – například při smrti hráčovy jednotky objekt zůstal stát na místě, což jsem obešel tím, že po smrti jednotky se spustí komponent, který objekt jednotky posouvá dozadu.

Dále byly na řadě souboje hráčových a nepřátelských jednotek. Jakmile hráčovy jednotky překročí bránu sekce úrovně a provede se matematická operace s hráčovými jednotkami, hra přejde do stavu konfrontace, kdy skupině hráčových jednotek je předána skupina nepřátelských jednotek a skupině nepřátelských jednotek je předána skupina hráčových jednotek. Jednotky v obou ze skupin si následně přiřadí nejbližší cíl, ke kterému se začnou pohybovat. Při kontaktu s nepřátelskou jednotkou obě jednotky zemřou. Hra nepokračuje dále, dokud jedna ze skupin není prázdná, přičemž jestliže je prázdná hráčova skupina, hra končí.

Jakmile hra byla hratelná, začal jsem dodělávat poslední detaily – animace, audio, monetizace.

Animace jsem získal ze stránky mixamo, protože animace obsažené v modelu postavy v dodaném asset packu neobsahovaly animaci smrti a kombinovat animace z mixamo a z modelu nebylo možné.

Pro přehrávání audia jsem vytvořil scriptable object **Sound**, který uchovává informace o audio clipu. **Sound** obsahuje nastavení pro rozsah výšek tónu, ze kterého **Audio Manager** vybírá náhodnou výšku tónu, aby z jednoho audio clipu mohl vytvořit několik variant a tím zajistil větší zvukovou rozmanitost. Žádost o přehrání audia, kterou zpracuje **Audio Manager**, se podává přes **AudioRequestEventChannel**, protože některé objekty, které přehrávají nějaký zvuk, nemohou na Audio Manager získat odkaz přímo.

Monetizaci jsem vyřešil balíčkem od Unity. Komponent **Monetization Manager** při startu hry načte reklamy, které se zobrazí kdykoliv po zavolání metody na **Monetization Manageru**.