# Dokumentace hry Last Man Standing

## Obsah

# 1 Vytvoření mapy

Čas: 2 h

Pro vytvoření mapy jsem používal výhradně Unity editor. Struktury ve hře jsou postavené pouze z jednoduchých kostek, které jsem různě otáčel nebo zvětšoval, abych vytvořil složitější tvary. Mapa je rozdělená do čtyř částí. V každé je jiná struktura, do které jsem později přidal i odměnu ve formě léčivého předmětu. Hráč se spawnuje na stupínku uprostřed mapy. Zpětně si uvědomuji, že na mapě jsem strávil asi dále, než by bylo pro tento projekt potřeba.

# 2 Input, pohyb hráče a pohyb kamery

Čas: 2 h

## 2.1 Input

*ClientInput, InputEvents*

Pohyb hráče je rozdělen to tří scriptů ClientInput, InputEvents a PlayerMovement. Hlavní myšlenka je v tom, aby registrování inputů nebylo přímo závislé na objektu hráče a později se mohly napojovat další mechaniky (například interagování s objekty pomocí myši). Další výhodou je to, že všechny inputy jsou v jednom scriptu.

ClientInput zaznamenává všechny inputy hráčem a jakmile nastane zmáčknutí klávesy spustí odpovídající metodu z InputEvents.

InputEvents se nachází mimo scénu a dědí z classy ScriptableObject. Po spuštění metody scriptem ClientInput se vyvolá odpovídající event. Scriptům, které chtějí reagovat na zmáčknutí klávesy, stačí přidat svou metodu do eventu v InputEvents. Protože se InputEvents nachází mimo scénu není závislý na žádném objektu.

## 2.2 Pohyb hráče

*PlayerMovement*

PlayerMovement řeší pohyb hráče. Ze scriptu InputEvents bere hodnoty a eventy pro pohyb jako například zmáčknutí mezerníku nebo WASD. Pohyb je pak vykonaný skrze metody rigidbody.velocity a rigidbody.addForce.

Dále jsem chtěl přidat otáčení kamery v menu kolem hráče. Následují dva řádky reagují na aktuální úhel otočení kamery a podle toho přepočítávají směr pohybu hráče, aby se hráč nepohyboval šikmo oproti kameře, ale rovně.

```
var moveDirectionX = inputEvents.verticalInput * Mathf.Sin(rotationInRad) +
inputEvents.horizontalInput * Mathf.Cos(rotationInRad);
var moveDirectionZ = inputEvents.verticalInput * Mathf.Cos(rotationInRad) -
inputEvents.horizontalInput * Mathf.Sin(rotationInRad);
```

Jednoduší řešení je otáčet celou mapou, ale to se mi zdálo zbytečně namáhavé na výkon, když stačí použít dva řádky kódu.

Vycházel jsem z následujících vzorců pro přepočet hodnot X a Y pro rotaci grafu o úhel alfa. Y´ a X´ jsou pak nové hodnoty v otočeném grafu.

X´ = x * Cos(alfa) – y * Sin(alfa)

Y´ = x * Sin(alfa) + y * Cos(alfa)

Vzorce jsem pak musel trochu upravit, protože počítají s tím, že úhel otáčení je proti směru hodinový ručiček a v unity je to po směru. Finální podoba v kódu je pak:

X´ = x * Sin(alfa) + z * Cos(alfa)

Z´ = x * Cos(alfa) – z * Sin(alfa)

Místo souřadnice Y používám Z. S reálnými hodnotami je to pak:

novýSměrXPoOtočení´ = původníSměrX * Sin(alfa) + původníSměrY * Cos(alfa)

novýSměrZPoOtočení ´ = původníSměrX * Cos(alfa) – původníSměrY * Sin(alfa)

Trvalo mi celkem dlouho, než jsem si uvědomil, že úhel otáčení je na druhou stranu. Myslím, že bez tohoto kroku jsem to mohl mít tak za 40 min;

## 2.3 Pohyb Kamery

*CameraMovement*

Nová pozice kamery se vypočítá podle aktuální pozice sledovaného objektu followedObject a počáteční pozice kamery, která udává vzdálenost od sledovaného objektu. Kód je psaný tak, aby šlo měnit hodnotu followedObject a kamera se tak pohybovala za novým cílem.

# 3 Základní nepřítel a spawnování nepřátel

Čas: 2 h

## 3.1 Pohyb nepřítele

*EnemyMovement*

Směr pohybu je směr k sledovanému objektu followedObjekt. Rychlost se pak přidá přes rigidbody.velocity. Výška y zůstává konstantní. Hodnota followedObjekt lze měnit pro pohyb za jiným objektem.

## 3.2 Spawání nepřátel

*InstanceManager, EnemySpawner, EnemyWave, EnemySpawn*

InstanceManager v sobě drží reference na všechny spawnuté nepřátele nebo hráče. Reference na InstanceManager jsou přes princip Singletonu.

EnemySpawn je typ, který v sobě drží hodnoty pro prefab nepřítele, počet nepřátel, kolik se má spawnout a interval po kterém se budou spawnovat. Když se na to zpětně koukám tak EnemySpawn nemusí být class ale měl by to být spíše struct.

EnemyWave je ScriptableObject, který slouží jako blueprint pro vytváření informace o určité vlně nepřátel. Každá nové vytvořená vlna se skládá z několika EnemySpawn.

EnemySpawner postupně čte informace z EnemyWave a podle toho spawnuje nepřátele přes metodu Instatiate. Lze nastavit i čas čekání mezi vlnami nebo před první vlnou. Nepřátelé se náhodně spawnují ve čtverci učeném dvěma body spawnPoint1 a spawnPoint2.

## 3.3 Timer

V tomto kroku jsem si také vytvořil pomocnou classu Timer. Která slouží jako časovač, než se stane určitý event. V mnoho případech by šel nahradit přes coroutine, ale radši používám tento způsob, protože lze lépe operovat se zbývajícím časem. Navíc mi přijde, že díky tomu napíšu méně kódu než s coroutine. Ale pro některé případy je určitě lepší využívat coroutine. Například v EnemySpawner bych musel pro stejný efekt využít vícekrát coroutine, ale stačí mi jen jeden Timer.

# 4 Střílení a projektily

Čas: 5 h

*Attack, EnemyAttack, PlayerAttack, ProjectileBehaviour*

Attack má v sobě logiku pro střílení projektilů. Buďto lze použít metodu ShootOnTargetPosition, která vystřelí projektil směrem na zadanou pozici nebo metodu Shoot, která vystřelí projektil směrem na pozici attackingOnObject.

EnemyAttack dědí z Attack. Pokud se přiblíží attackingOnObject do určité vzdálenosti začne střílet směrem na něj.

PlayerAttack dědí z Attack. Logika je rozdělena do dvou částí. Pokud hráč drží tlačítko na myši je každý update zavolána metoda ManualAttack. Ta přepočítá pozici myši na obrazovce na pozici ve světě a s tímto parametrem zavolá metodu ShootOnTargetPosition. Druhá část je AutomaticAttack, když hráč nedrží tlačítko na myši. Ta zavolá metodu Shoot pokud se attackingOnObject přiblíží do určité vzdálenosti.

ProjectileBehaviour script je na každém projektilu a určuje jeho chování. V tomto kroku jsem nastavil, aby se po určité době zničil nebo když trefí objekt ve vybrané vrstvě tak se také zničí.

Nejvíce času jsem v strávil na práci v Inspektoru. Vytváření prefabu, zadávání hodnot a testování.

# 5 Životy hráče a nepřátel

Čas: 30 min

*IHpSystem, PlayerHp, EnemyHp*

IHpSystem je interface s metodou Damage. ProjectileBehaviour pak využívá tuto metodu u objektů které dědí z IHpSystem.

PlayerHp, EnemyHp jsou jednoduché classy na počítání životů hráče a nepřátel. Definují metodu Damage. Metody PlayerDied a EnemyDied se zavolají, když životy spadnou na nulu.

# 6 Update hledání nepřátel

Čas: 40 min

*PlayerAttack, PlayerMovement*

U *PlayerMovement* byla zaměněna metoda Physics.OverlapBox za Physics.OverlapBoxNonAlloc. U *PlayerAttack* se dříve při hledání nejbližšího nepřítele iterovalo přes všechny nepřátele. Nyní se využívá metoda Physics.OverlapSphereNonAlloc pro deset nepřátel v určitém okruhu. Tyto změny sníží náročnost hry na výkon.

# 7 Varianty nepřátel

Čas: 40 min

*EnemyMovementDodger*

EnemyMovementDodger dědí z EnemyMovement. Upravuje pohyb nepřátele, aby se pohyboval po sinusové trajektorii na ose Y.

```
var velocityOnYAxis = heightMaxOffset * movementSpeedOnYAxis *
Mathf.Cos(timeForSinusFunc * movementSpeedOnYAxis);
```

Rychlost pohybu vypočítávám přes následující vzorec

rychlost Y = amplituda * rychlost * cos (t * rychlost)

Je to derivovaný vztah ze základního vzorce. Navíc sem dávám, jaké vzorce bych používal v jiných metodách.

pozice Y = amplituda * sin (t * rychlost) => Transform.Translate()

rychlost Y = amplituda * rychlost * cos (t * rychlost) => rigidBody.velocity()

zrychlení Y = amplituda * $rychlost^2$ * - sin (t * rychlost) => rigidBody.addForce()

# 8 Spawnpointy

Čas: 5 min

*EnemySpawner*

Místo dvou bodů se do EnemySpawner v inspektoru přiřadí pozice různých spawnPointů, které se rozmístí po mapě. EnemySpawner při spawnování nepřítele pak jeden náhodně vybere a nepřítel se spawne na něm.

# 9 UI a GameLoop

Čas: 2 h

*MenuManager, GameEnder, GameInfo, MouseTracker*

MenuManager se stará o přechod z menu do hry a obráceně. Metodu pro spuštení hry StartGame volá tlačíko Play v menu. Zavolaná metoda upraví UI pro Hru a změní stav hry v Game Info. Podobně pak funguje metoda GoToMainMenu.

GameEnder obsahuje metodu RestartGame, která je zavolána při smrti hráče. RestartGame pak vrací scénu do původního stavu. Vymaže všechny spawnuté nepřátele a spawne nového hráče, kterému přiřadí hodnoty o UI, aby se mohli správně zobrazovat životy hráče. Poté zavolá metodu GoToMainMenu ze scriptu MenuManager.

GameInfo v sobě drží informace o stavu hry jako například jestli je hra v menu nebo jestli se hraje. Je to ScriptableObject takže se nachází mimo scénu. Potenciálně v budoucnu lze nahrát jiný GameInfo script s jinými hodnotami o hře pro testování nebo jiný herní mód.

Do scriptu MouseTracker jsem přesunul logiku z PlayerAttack o hledání pozice myši, aby tuto funkci mohly využívat i ostatní scripty.

Nejvíce času jsem strávil v unity inspektoru při vytváření UI.

# 10 Počítadlo Zabití

Čas: 2 h

*KillCounter*

KillCounter zobrazuje v UI počet zabití jednotlivých nepřátel. Statistiky uchovává v dictionary killStatistics. Pokud nepřítel umře zavolá metodu Died se svým jménem. Ta zkontroluje, jestli už byl daný nepřítel zabit. Jestli ano pouze přičte zabití a aktualizuje UI.  Pokud ne spawne nový komponent řádky jako child k určenému místu v UI. Metoda Reset vymaže všechny hodnoty z killStatistics a vytvořené objekty řádek v UI. Ta se pak volá ze scriptu GameEnder.

# 11 Pohyb kamery v menu

Čas: 1.5 h

*CameraMovement*

Do scriptu CameraMovement jsem přidal pohyb kamery, když je v menu. Tam se otáčí kolem hráče. Výsledný úhel otočení kamery je pak zapsán do GameInfo, aby s ním mohl pracovat script PlayerMovement.

Na začátku se zavolá metoda CalculateStartingValues, která vypočítá startovní hodnoty, tak aby se z dané pozice otáčela do kruhu kolem středu. Kameru lze tak umístit kdekoliv na scéně a vždy se bude otáčet kolem středu mapy a nebude měnit úhel, s který se kouká na střed.

# 12 Efekty (Particles)

Čas: 1 h

*HpSystem*

Interface IHpSystem jsem změnil na classu HpSystem, protože classy PlayerHp a EnemyHp, které z něho dědí mají více společných věcí a nechtěl jsem duplikovat kód. Logiku Particles jsem pak jednoduše zakomponoval do PlayerHp a EnemyHp, přes děděné metody Hit a Died.

# 13 Léčivý předmět

Čas: 30 min

*HealItem, HpSystem*

HealItem zajišťuje, že když do objektu narazí jiný objekt z vybrané vrstvy zavolá na něm metodu Heal ze scriptu HpSystem, která přičte životy. Poté se objekt zničí.

Když nepřítel umře vytvoří objekt HealItem na svém místě. Logika je vložena do metody Died ze scriptu EnemyHp.

# 14 Update počítadlo zabití pomocí coroutine

Čas: 10 min

*KillCounter*

Do scriptu KillCounter jsem přidal logiku, která se zavolá každé tři vteřiny a zkontroluje, jestli se hodnoty změnily a jestli ano tak aktualizuje UI. Využil jsem k tomu coroutine. Vše jsem ale dal do komentářů, protože je to v mém případě zbytečné. UI se aktualizuje hned po tom co se hodnoty změní, proto je tato kontrola zbytečná.

# 15 Assert methoda a kontrola souborů

Čas: 3 h

V tomto kroku jsem prošel všechny scripty a do Awake metod jsem přidal Assert všude tam, kde se něco zadává přes inspektor. U některých scriptů jsem změnil názvy proměnných, aby lépe reprezentovali to, co znamenají.

O Assert jsem dříve nevěděl a myslím, že bych to asi využíval už od začátku, kdybych s tím měl zkušenosti.

# 16 Počítadlo zabití

Čas: 10 min

Ke konci jsem zjistil, že jsem počítadlo zabití neudělal podle zadání, tak jsem ho musel trochu poupravit. Naštěstí to stačilo pouze zjednodušit a věci jsem spíše mazal, než vytvářel.

# 17 Dokumentace

Čas 4.5 h