Vyšší odborná škola a Střední průmyslová škola, Šumperk,
Gen. Krátkého 1, 787 29 Šumperk

Obor vzdělání: Informační technologie (18-20-M/01)

MATURITNÍ PRÁCE

z odborných předmětů

Školní rok: 2018/2019 **Jakub Píšek**
Třída: IT4A

Vyšší odborná škola a Střední průmyslová škola, Šumperk,
Gen. Krátkého 1, 787 29 Šumperk

Obor vzdělání: Informační technologie (18-20-M/01)

ZADÁNÍ MATURITNÍ PRÁCE

z odborných předmětů

Žák: **Jakub Píšek**

Téma maturitní práce: **Návrh a vývoj programu**

Obsah maturitní práce:
1. Návrh a objektový návrh
2. Návrh a tvorba GUI
3. Vývoj programu
4. Testování a odladění
5. Tvorba dokumentace

Další náležitosti zadání: **K:/sablony/maturitní práce**

Datum zadání: **27. září 2018**

Datum odevzdání: **29. března 2019**

Vedoucí práce: **Mgr. Zdeněk Přikryl** **Podpis:**

PROHLÁŠENÍ

Prohlašuji, že jsem maturitní práci z odborných předmětů vypracoval (a) samostatně a k vypracování jsem použil (a) pouze zdroje uvedené v seznamu literatury.

Datum: 28. března 2019 **Žák: Jakub Píšek**

Podpis:

Vyšší odborná škola a Střední průmyslová škola, Šumperk,
Gen. Krátkého 1, 787 29 Šumperk

KONZULTAČNÍ LIST

Žák: Jakub Píšek **Třída: IT4A**

Téma maturitní práce: Návrh a vývoj programu

Datum kontroly: 15. 11. 2018

Obsah kontroly: Návrh a objektový návrh

Hodnocení kontroly: 1 2 3 4 5 Podpis vedoucího práce: _____

Datum kontroly: 20. 12. 2018

Obsah kontroly: Přípravné práce a GUI

Hodnocení kontroly: 1 2 3 4 5 Podpis vedoucího práce: _____

Datum kontroly: 24. 1. 2019

Obsah kontroly: Vývoj programu

Hodnocení kontroly: 1 2 3 4 5 Podpis vedoucího práce: _____

Datum kontroly: 14. 2. 2019

Obsah kontroly: Vývoj programu

Hodnocení kontroly: 1 2 3 4 5 Podpis vedoucího práce: _____

Datum kontroly: 7. 3. 2019

Obsah kontroly: Zpracování dokumentace

Hodnocení kontroly: 1 2 3 4 5 Podpis vedoucího práce: _____

OBSAH

1 ÚVOD

Videohra naprogramovaná v herním enginu Unity (2D) v programovacím jazyce C#. Použité vývojové prostředí Visual Studio 2017.

Hra je navržena jako simulace běžného života, kdy se musíte starat o svou postavičku jako o sebe samého. Musíte si hlídat hladinu základních lidských potřeb a uspokojovat je.

Za cíl je vytvořit lehce škálovatelnou hru, což znamená, že se dá bez větších obtíží dále rozšiřovat. Kód bude přehledný a stručně popsaný, aby se v něm mohl lehce zorientovat i úplný nováček ve tvorbě her.

Motivací bylo naučit se něco nového, rozšířit si obzory programování o nový programovací jazyk C#, nové postupy a logiky práce s objektově orientovaným programováním. Najít novou zálibu v programování a nahlédnout za oponu tvorby her. Dalším pohonem bylo se naučit grafický design – PixelArt v tomto případě. Velmi nenáročný styl kresby na výkon počítače, tudíž možností je nespočet, co vše se dá integrovat.

Obrázek 3 - Unity logo

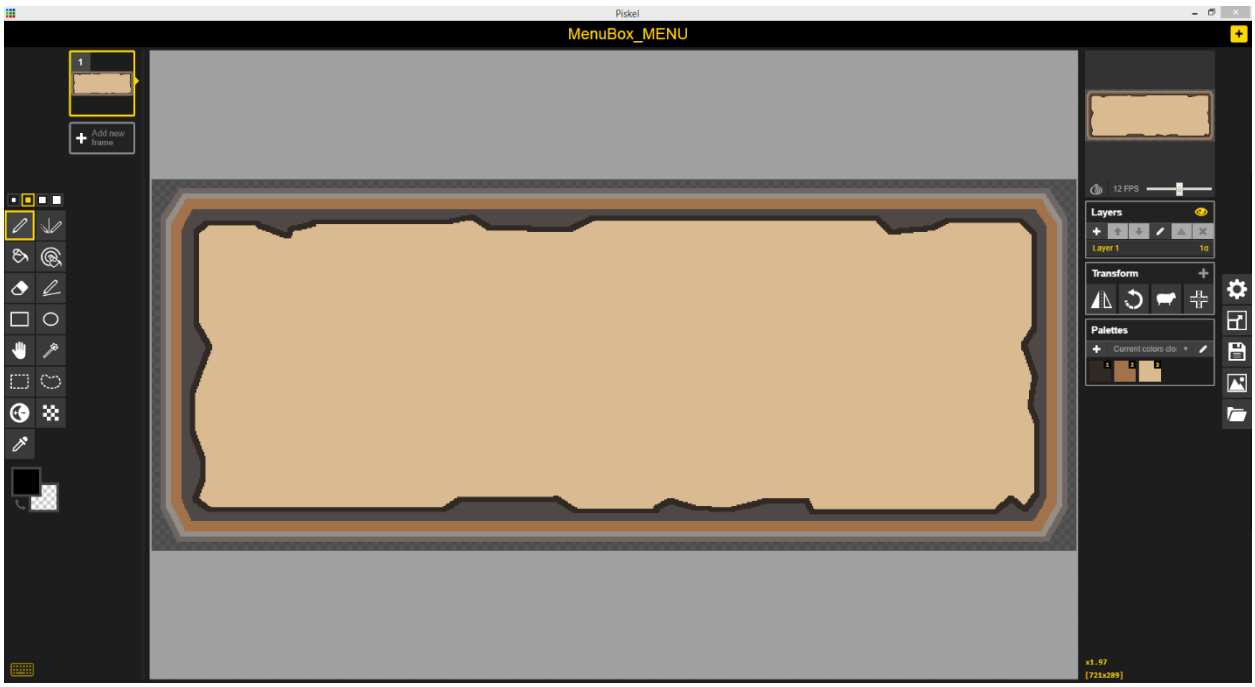
Obrázek 2 - C# logo

Obrázek 1 - Visual Studio 2017

2 POUŽITÉ TECHNOLOGIE

2.1 Piskel

Bezplatný grafický editor pro PixelArt a jeho animace. Je možnost pracovat online přímo v prohlížeči nebo si z oficiální stránek stáhnout offline verzi programu, což je můj případ. Program není třeba instalovat, stačí stáhnout ZIP soubor, ten rozbalit a spustit exe soubor. Pracoval jsem na verzi Piskel-0.14.0.



Obrázek 4 - Piskel ukázka

Výhody:

- Zdarma, open source
- Přehledné a jednoduché rozhraní
- Vhodné pro začátečníky
- Import obrázků
- Zobrazí náhledu animací (1 fps – 24 fps)
- Možnost exportu obrázků jako animaci GIF
- Škálování rozlišení

Nevýhody:

- Změna barev v paletě neovlivní barvy na obrázku
- Nedostupná automatická aktualizace programu
- Chybí posouvání po stránce pomocí držení kolečka myši
- Špatná kompatibilita s grafickým tabletem

2.2 Unity

Multiplatformní herní engine od společnosti Unity Technologies. Umožňuje vývoj 3D a 2D her jak na PC, tak i na konzole a chytré mobilní telefony. Engine nabízí grafické rozhraní a podporu tvorby skriptů v jazyce C# a Javascript. Samotné Unity bylo naprogramováno v jazycích C++ a C#.

Mezi nejznámější tituly, které věží na Unity enginu, patří Ori and the Blind Forest, Cuphead, Rust, Kerbal Space Program, Hearthstone: Heroes of Warcraft a Temple Run Trilogy.

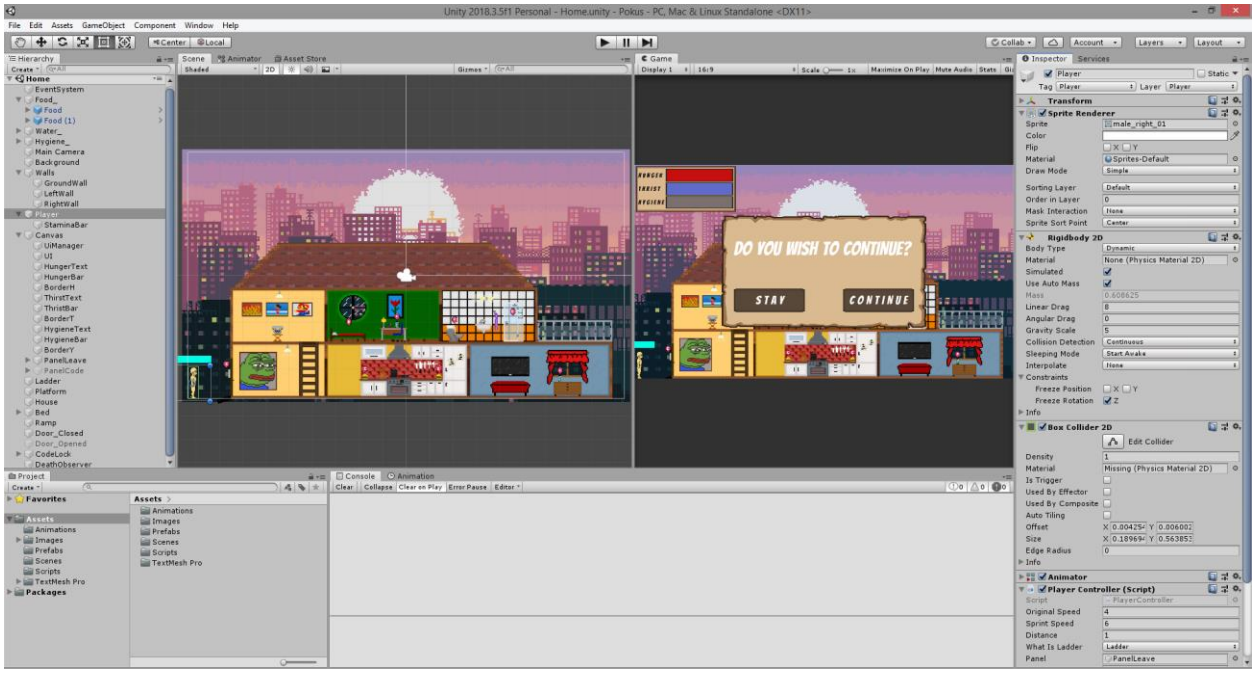
Unity si nebere žádnou část ze zisku prodaných kopií her vytvořených skrz tento engine. Všechno, co v něm vytvoříte, je jen a jen vaše.

Přestože Unity není open source, podporuje komunitní obsah skrz svůj Asset Store https://assetstore.unity.com [1], kde uživatelé mohou zveřejňovat své dodatky do Unity, většinou grafické, které mohou i zpoplatnit.

Literatura a video návody Unity jsou zcela zdarma pro každého na oficiálních stránkách https://docs.unity3d.com/Manual/. [2]

Pracoval jsem s licencí Personal, která dovoluje bezplatné užívání k osobním účelům nebo při hrubých ročních ziscích nepřekračující hranici 100 000$. Více k licenci zde: https://store.unity.com/products/unity-personal. [3]

Hra byla vyvíjena na různých verzích vydaných v roce 2018, ale finálně běží na verzi 2018.3.5f1.



Obrázek 5 - Unity ukázka

Výhody:

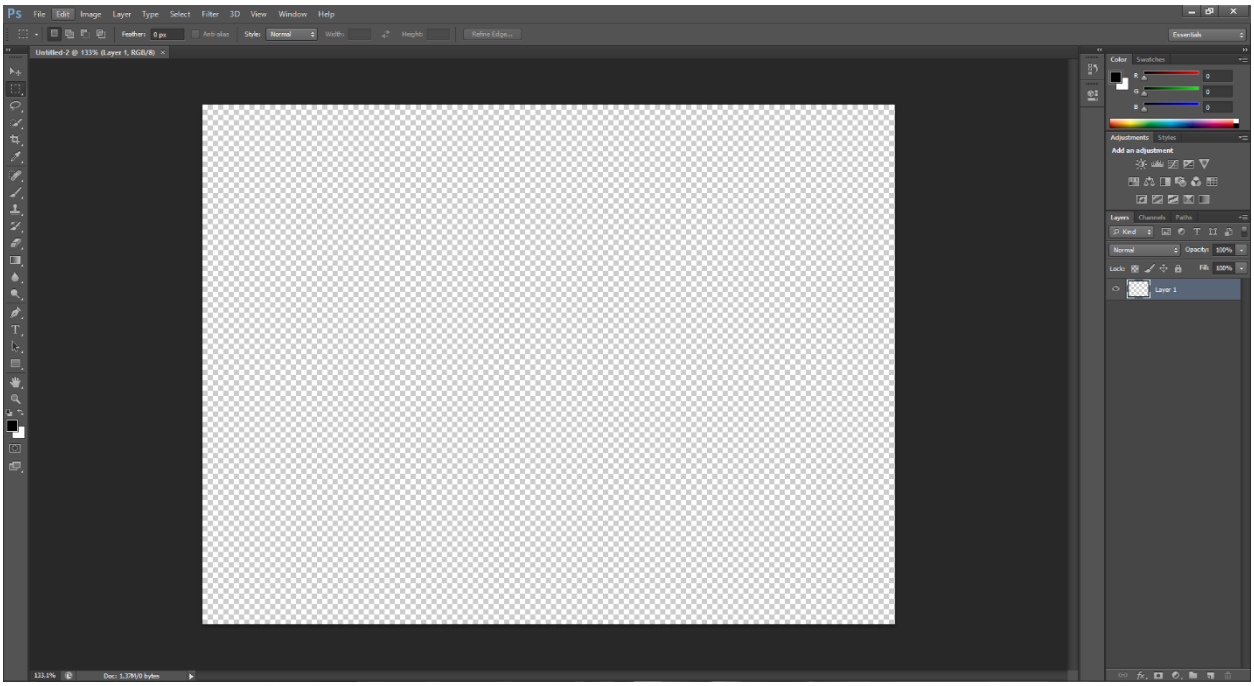
- Zdarma pro výukové účely
- Snadná orientace v rozhraní
- Rozsáhlé nástroje pro práci s objekty
- Multiplatformní
- Jednoduchý přístup z jednoho objektu do jiného
- Nejlepší 2D engine na trhu

Nevýhody:

- Nevhodné pro AAA tituly
- Některé dodatky zbytečně drahé
- Zaostává oproti jiným enginům v 3D odvětví

2.3 Photoshop

Komerční bitmapový grafický editor pro veškerou bitmapovou grafiku, vyvíjena společností Adobe Systems. Program lze zakoupit na oficiálních stránkách https://www.photoshop.com. [4] V programu se pracovalo na školních počítačích, kde je zakoupena licence. Ve Photoshopu pro mne grafiku vytvořil zdarma můj spolužák Patrik Schneider.



Obrázek 6 - Photoshop ukázka

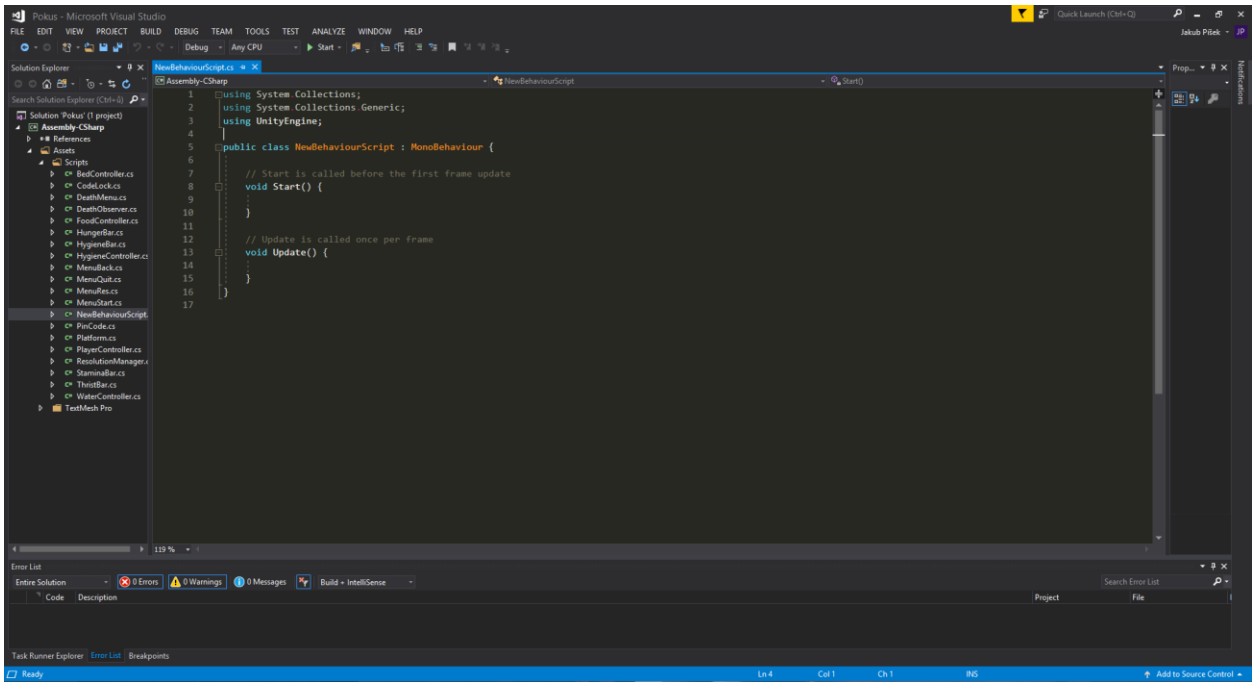
2.4 Visual Studio 2017

Vývojové prostředí (IDE) pro Windows, iOS, Android a jiné od firmy Microsoft. Používá se jak pro konzolové aplikace, tak i pro aplikace s grafickým rozhraním. Může být použito pro vývoj konzolových aplikací a aplikací s grafickým rozhraním spolu s aplikacemi Windows Forms, webovými stránkami, webovými aplikacemi a webovými službami jak ve strojovém kódu, tak v řízeném kódu na platformách Microsoft Windows, Windows Mobile, Windows CE, .NET, .NET Compact Framework a Microsoft Silverlight.

Visual Studio obsahuje editor kódu podporující IntelliSense a refaktorování. Integrovaný debugger pracuje jak na úrovni kódu, tak na úrovni stroje. Další vestavěné nástroje zahrnují designer formulářů pro tvorbu aplikací s GUI, designer webu, tříd a databázových schémat.

Visual Studio podporuje jazyky prostřednictvím jazykových služeb, což umožňuje, aby editor kódu a debugger podporoval jakýkoliv programovací jazyk. Mezi vestavěné jazyky patří C/C++ (použitím Visual C++), VB.NET(použitím Visual Basic .NET) a C# (použitím Visual C#). Podpora dalších jazyků jako Oxygene, F#, Python a Ruby spolu s ostatními může být přidána jazykovými službami, které musí být nainstalovány zvlášť. Také je podporováno XML/XSLT, HTML/XHTML, JavaScript a CSS. [5]

Pracoval jsem s licencí Community, která je dostupná zdarma a neúčtuje si žádné poplatky.



Obrázek 7 - Visual Studio ukázka

3 NÁVRH PROGRAMU

Tématem je simulace běžného života člověka

3.1 Styl

1) Simulátor
- Realistická hra
- Platí stejné zákony jako v realitě

2) Grafika
- PixelArt
- Druh grafiky, která je vytvářená po jednotlivých pixelech
- Velmi často užívaný k tvorbě 2D her

3) Přežití
- Máte za úkol se o postavu starat
- Pokud základní potřeby zanedbáte, může to mít negativní dopad na jeho život
- Maximální penalizace – smrt, konec hry

3.2 Scény

1) Menu
- Rozcestník programu
- Tlačítko pro spuštění
- Tlačítko s výběrem rozlišení
- Tlačítko pro ukončení hry

2) Domov
- Scéna domu, ve kterém hráč bydlí
- Za úkol má se najíst, udělat hygienu
- Uspokojit své potřeby (jídlo, pití atd.)

3) Smrt
- Výpis důvodu smrti
- Restart úrovně
- Návrat do hlavního menu

3.3 Hlavní postava

3.3.1 Atributy

- Hlad
- Žízeň
- Hygiena
- Pohyb
- Rychlost

- Všechny atributy typu Float
- Změna hodnoty jednoho atributu může ovlivňovat hodnotu druhého atributu
  - Pokud bude se hráč nají, zašpiní se
  - Rychlejší chůze dehydratuje atd.

3.3.2 Funkce

1) Pohyb

- Ovládá se pomocí WASD
- Při kolizi s některými objekty bude možnost interakce (sebrání, konzumace)
- Samotné ovládání bude ve hře jako Nápověda, kde bude vše ukázáno

2) Smrt

- Ukončení levelu
- Nastavení defaultních hodnot
- Dialogové okno s nabídkou možností – restart, návrat do menu

3.4 Objekty

1) Jídlo

- Konzumovatelný předmět
- Zažene potřebu hladu

2) Pití

- Konzumovatelný předmět
- Zažene žízeň

3) Záchod, sprcha

- Možná interakce
- Zažene potřebu hygieny

4 OBJEKTOVÝ NÁVRH

4.1 Player

Hlavní postava, za kterou ve hře hrajete. Na objektu je připojeno několik komponentů Unity a C# skript. Obsahuje i jeden přidružený objekt StaminaBar – ukazovat výdrže, který se pohybuje s postavou po scéně.

4.1.1 Komponenty

1) Transform
- Určení pozice, rotace, velikosti objektu

2) Sprite Renderer
- Renderovaní objektu, jak se bude zobrazovat ve scéně

3) Rigidbody 2D
- Komponent zajišťující fyziku 2D objektů

4) Box Collider 2D
- Neviditelný tvar, který obstarává kolizi objektu ve 2D

5) Animator
- Přiřazuje objektu animace

4.1.2 Skript PlayerController

4.1.2.1 Proměnné

- **Private**
- movementSpeed
    - typ float
    - rychlost pohybu
- originalSpeed
    - typ float
    - SerializeField – zviditelní proměnnou v editoru Unity
    - násobitel rychlosti pohybu při chůzi
- sprintSpeed
    - typ float
    - SerializeField – zviditelní proměnnou v editoru Unity
    - násobitel rychlosti pohybu při sprintu

- moveHorizontal
    - typ float
    - nabírá horizontální hodnotu ze vstupu (A, D, ←, →)
- moveVertical
    - typ float
    - nabírá vertikální hodnotu ze vstupu (W, S, ↑, ↓)
- isClimbing, isFalling
    - typ boolean
    - kontroluje, zda hráč leze po žebříku
- deathReason
    - typ string
    - obsahuje důvod hráčovy smrti

- **Public**

- distance
    - typ float
    - hodnota délky paprsku, který kontroluje kolizi se žebříkem
- whatIsLadder
    - typ LayerMask, proměnná UnityEngine
    - upřesňuje vrstvu, kterou má použít funkce Physics.Raycast

- **Static**

- maxHunger
    - typ float
    - maximální dosažitelná hodnota hladu
- maxThirst
    - typ float
    - maximální dosažitelná hodnota žízně
- maxHygiene
    - typ float
    - maximální dosažitelná hodnota hygieny
- maxStamina
    - typ float
    - maximální dosažitelná hodnota výdrže
- hunger, thirst, hygiene, stamina
    - všechny typ float
    - uchovávají aktuální hodnoty hladu, žízně, hygieny a výdrže

- **Private komponenty**

- rb
    - reference na Rigidbody2D na objektu

- sR
    - reference na SpriteRenderer na objektu

- **Public komponenty**

- panel
    - reference na GameObject ve scéně
    - panel PanelLeave v Canvasu
- animator
    - reference na Animator na objektu
- dO
    - reference na skript DeathObserver jiného objektu

4.1.2.2 Metody

1) void Start

- Metoda Unity, která se zavolá vždy při vytvoření objektu, na který je skript vázán.

```
39      void Start () {
40          // Načtení komponentů objektu
41          rb = GetComponent<Rigidbody2D>();
42          sR = GetComponent<SpriteRenderer>();
43          animator = GetComponent<Animator>();
44
45          // Skrytí panelu
46          panel.SetActive(false);
47
48          // Nastavení defaultních hodnot
49          SetOriginalStats();
50      }
```

Obrázek 8 - PlayerController, metoda Start

Metoda Start, typu void, je vhodná pro dosazení komponentů do proměnných. Taktéž je dobrá pro nastavení hodnot při prvním vykreslení scény.

V metodě Start jsem si vložil komponenty do proměnných, abych s nimi mohl dále jednoduše pracovat a přistupovat k jejich funkcím.

Dále jsem skryl panel **PanelLeave**, který by jinak zavazel a nepustil by hráče dál. Metoda **SetOriginalStats** nastaví vybraným proměnným původní hodnoty.

Obrázek 9 – Scéna Home, panel PanelLeave ukázka

```
158     // Metoda pro nastavení proměnných na původní hodnoty
159     public void SetOriginalStats () {
160         thirst = 0;
161         hunger = 0;
162         hygiene = 0;
163         stamina = maxStamina;
164         movementSpeed = originalSpeed;
165     }
```

Obrázek 10 - PlayerController, metoda SetOriginalStats

2) <u>void Update</u>

- Metoda Unity, typu void, která se volá při vykreslování každého snímku, pokud je objekt, na který je skript vázán, povolen. Čas mezi voláním metody se může lišit.

```
53  void Update () {
54      // Přičítání potřeb
55      if (hunger < maxHunger) {
56          hunger += 1.0f * Time.deltaTime;
57      }
58
59      if (thirst < maxThirst) {
60          thirst += 0.8f * Time.deltaTime;
61      }
62
63      if (hygiene < maxHygiene) {
64          hygiene += 0.5f * Time.deltaTime;
65      }
66
67      // Kontrola smrti
68      if (hunger >= maxHunger || thirst >= maxThirst || hygiene >= maxHygiene) {
69          Die();
70      }
71
72  }
```

Obrázek 11 - PlayerController, metoda Update

V metodě Update se nachází několik podmínek typu **if**, které kontrolují, zda aktuální hodnota proměnných nepřesáhla maximální možnou hodnotu. Pokud první tři podmínky vrátí hodnotu true, v proměnných se začne zvětšovat hodnota daným číslem * **Time.deltaTime**. Funkce Unity Time.deltaTime vrací hodnotu času, který uběhl mezi jednotlivými snímky. Tato funkce je velmi užitečná v případech, kdy hra běží na počítačích s různými komponenty a mezi nimi se snímky za sekundu mohou lišit, a to by způsobovalo rozdílné chování.

Čtvrtá podmínka kontroluje, zda došlo k dorovnání, či přesažení maximálních hodnot. Porovnává se zároveň více hodnot pomocí logického operátoru **OR** (| |), který vrátí hodnotu true tehdy, kdy je alespoň jeden z operandů **true**.

Pokud podmínka vrátí hodnotu true, spustí se metoda Die, která řeší hráčovu smrt – konec hry.

```
168     // Metoda řešící smrt hráče
169     public void Die () {
170         // Zjištění důvodu smrti
171         if (hunger >= maxHunger) {
172             deathReason = "Hunger";
173         }
174         else if(thirst >= maxThirst) {
175             deathReason = "Thirst";
176         }
177         else if(hygiene >= maxHygiene) {
178             deathReason = "Hygiene";
179         }
180 
181         // Předání parametru do metody jiného objektu
182         dO.SetObserver(deathReason);
183         // Zavolání metody a načtení scény
184         SetOriginalStats();
185         SceneManager.LoadScene("Death");
186     }
```

Obrázek 12 - PlayerController, metoda Die

Metoda pomocí větvených podmínek **if** a **else if** zjistí příčinu smrti – pokud jedna z proměnných dosáhla maximální hodnoty. Do stringu **deathReason** se načte řetězec znaků s důvodem smrti. Ten se poté skrz odkaz na skript **dO** načte do **setteru** SetObserver jako parametr. Potom se zavolá již zmiňovaná metoda SetOriginalStats a pomocí **SceneManageru**, funkcí Unity, se načte scéna Smrt.

3) <u>void FixedUpdate</u>

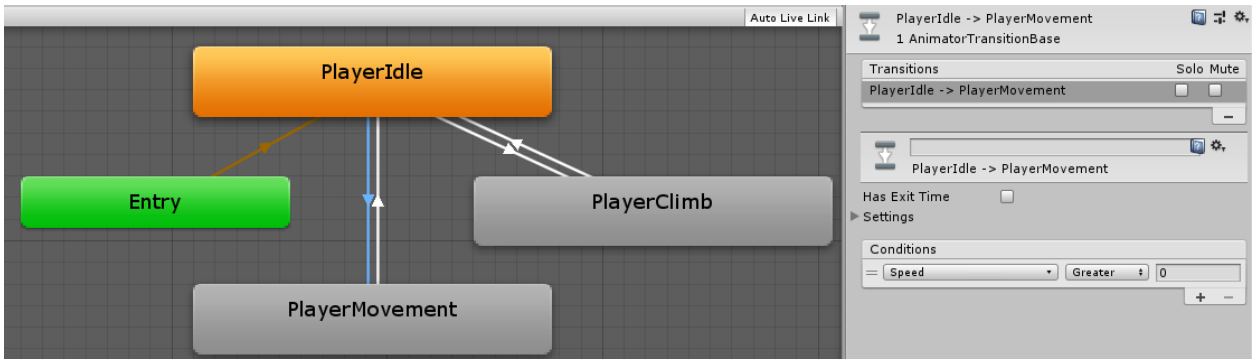
- Metoda Unity, typu void, která se volá před každou fyzikální změnou (pohyb objektů vlivem fyziky apod.). Defaultně se volá každé 0.2 sekundy a čas je vždy konstantní.

```
73      // FixedUpdate lépe řeší fyziku - pohyb, gravitaci, výstřely
74      void FixedUpdate () {
75 
76          #region MOVEMENT
77          // Načítání hodnot z klávesnice - horizontální input (A, D, ←, →)
78          moveHorizontal = Input.GetAxisRaw("Horizontal");
79 
80          // Přetočení postavy podle směru chůze
81          if (moveHorizontal < 0) {
82              sR.flipX = true;
83          }
84          else if (moveHorizontal > 0) {
85              sR.flipX = false;
86          }
87 
88          // Animace
89          animator.SetFloat("Speed", Mathf.Abs(moveHorizontal));
90 
91          // Předání hodnot do Rigidbody2D
92          Vector2 movement = new Vector2(moveHorizontal * movementSpeed, rb.velocity.y);
93          rb.velocity = movement;
94          #endregion
```

Obrázek 13 - PlayerController, metoda FixedUpdate, snímek 1/3, region Movement

V metodě FixedUpdate se nachází **region** Movement (blok kódu, přehlednost) pro pohyb postavy. Zjišťuje se zde input z klávesnice, v tomto případě horizontální, a ten se ukládá do proměnné **moveHorizontal**. Funkce **GetAxisRaw** nabírá hodnot -1, 0, 1 podle tlačítek A, nic, D. Dále je zde řešeno přetočení vykreslení hráče pomocí odkazu na komponent **sR** skrze podmínky **if** a **else if**. Defaultně je kresba hráče otočena doprava, takže pokud první podmínka vrátí true, což by znamenal pohyb doleva, kresba se na ose x přetočí na 180°. Pokud vrátí true druhá podmínka, přetočení se vrátí na 0°, defaultní hodnotu.

Řádek 89 nastavuje v komponentu Animator hodnotu Speed, typu float. Matematická funkce **Mathf.Abs** převádí hodnoty proměnných v parametru na absolutní hodnoty, což je vždy kladné číslo. To poté slouží ke správnému fungování animací.



Obrázek 14 - Animator, animace Playera

A nakonec se inicializuje **Vector2**, funkce UnityEngine, která do parametrů nabírá x a y vektory. Vektory se mohou modifikovat i uvnitř parametrů. Do vektoru x se načítá momentální směr chůze * **movementSpeed**. Y vektor načítá hodnoty z komponentu **Rigidbody2D**. Tento Vector2 se poté předá do odkazu **rb** jako rychlost po osách x a y.

```
96      #region SPRINT
97      // Podmínka - zda hráč drží Levý Shift a pohybuje se
98      if (Input.GetKey(KeyCode.LeftShift) && stamina > 0 && moveHorizontal != 0 && isClimbing != true) {
99          movementSpeed = sprintSpeed;
100         stamina -= 10.0f * Time.deltaTime;
101         thrist += 5.0f * Time.deltaTime;
102     }
103     else {
104         movementSpeed = originalSpeed;
105         // Stamina se doplňuje, pokud hráč neběží a stamina není plná
106         if (stamina < maxStamina) {
107             stamina += Time.deltaTime;
108         }
109     }
110
111     if (stamina <= 0) {
112         movementSpeed = originalSpeed;
113     }
114     #endregion
```

Obrázek 15 - PlayerController, metoda FixedUpdate, snímek 2/3, region Sprint

Druhým regionem v metodě je Sprint, který řeší zrychlení chůze hráče po zaznamenání klávesy. Jako první se zde nachází podmínka **if**, která vrátí hodnotu true právě tehdy, kdy podle logického operátoru **AND** (&&) vrátí každý operand true. To znamená, že na klávesnici musí být stisknutý **levý Shift**, zároveň musí proměnná **stamina** větší než nula, hráč se musí pohybovat a boolean **isClimbing** se nesmí rovnat true. V podmínce se stane, že se do násobitele rychlosti chůze vloží hodnota rychlosti sprintu, což zrychlí hráčův pohyb po scéně. Proměnná stamina se začne každým zavoláním metody FixedUpdate snižovat a proměnná **thirst** se násobí větší hodnotou, než jakou se násobí v metodě Update. Pokud podmínka vrátí **false**, násobiteli rychlosti se přiřadí defaultní hodnota a proběhne vnořená podmínka **if**, která kontroluje, zda je stamina menší než maximální hodnota, pokud vrátí **true**, **stamina** se začne každým zavoláním metody regenerovat. Nakonec se zde nachází podmínka taktéž **if** a ta ověřuje, zda je hodnota proměnné stamina menší nebo rovna nula. Pokud ano, násobitel rychlosti se taktéž vrátí na defaultní hodnotu.

```
118     #region LADDER
119     // Vystřelení neviditelného paprsku
120     RaycastHit2D hitInfo = Physics2D.Raycast(transform.position, Vector2.up, distance, whatIsLadder);
121
122     // Kontrola kolize paprsků Raycast
123     if (hitInfo.collider != null) {
124         if (Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.S)) {
125             isClimbing = true;
126         }
127         else {
128             // Pokud se zaznamená horizontální input
129             if (Input.GetKeyDown(KeyCode.A) || Input.GetKeyDown(KeyCode.D)) {
130                 isClimbing = false;
131             }
132         }
133     }
134
135     // Zjistění, jestli hráč padá
136     if (rb.velocity.y <= - 1) {
137         isFalling = true;
138     }
139     else if (rb.velocity.y > - 1) {
140         isFalling = false;
141     }
```

Obrázek 16 - PlayerController, metoda FixedUpdate, snímek 2/3, region Ladder snímek 1/2

Třetím a posledním regionem v metodě FixedUpdate je Ladder, řeší zaznamenávání a pohyb po žebříku. Je zde funkce **RaycastHit2D** z UnityEngine, která vrací informace o objektu, který detekovala paprskem z **Physics2D.Raycast**. Funkce nabírá až 6 parametrů, v mém případě mi stačí využít jen první 4. Zdroj, směr, délka a vrstva detekce paprsku. Tudíž pozice hráče, směr nahoru, dosah paprsku a vrstva jsou určeny v Unity editoru.

Můžete si všimnout i privátních proměnných originalSpeed a sprintSpeed, které jsou typu Unity.SerializeField, což zpřístupní proměnné Unity editoru. Pro ostatní třídy však tato proměnná zůstane privátní a nemohou k ní přistupovat.

Obrázek 17 - PlayerController, Unity editor

Podmínka **if** na řádku 123 ošetřuje, zda **Raycast** koliduje s jiným objektem. Dále následuje vnořená podmínka, opět typu **if**, která kontroluje vertikální vstup z klávesnice. Opět podle logického operátoru **OR**. Pokud se vyhodnotí jako **true**, do booleanu **isClimbing** se přiřadí hodnota true. Jestli podmínka vrátí **false**, provede se **else**, kde se nachází vnořená podmínka **if**, která kontroluje horizontální vstup. Pokud vrátí **true**, **isClimbing** se přiřadí hodnota false.

Kousek níž se nachází větvená podmínka **else if**, která kontroluje hráčovu rychlost na ose **y**. Pokud se první podmínka **if** vyhodnotí jako true, pak se **isFalling** nastaví na true. Jestliže se vyhodnotí jako true až druhá podmínka **else if**, potom **isFalling** nabyde hodnoty false. Tato podmínka pomáhá určit, zda hráč padá či nikoli.

```
143         // Pohyb po žebříku
144         if (isClimbing == true && hitInfo.collider != null) {
145             // Načítání hodnot z klávesnice - vertikální input (W, S, ↑, ↓)
146             moveVertical = Input.GetAxisRaw("Vertical");
147             // Předání hodnot do Rigidbody2D
148             rb.velocity = new Vector2(rb.velocity.x, moveVertical * movementSpeed);
149             rb.gravityScale = 0;
150             // Animace
151             animator.SetBool("IsClimbing", true);
152         }
153         else {
154             rb.gravityScale = 5;
155             isClimbing = false;
156             // Animace
157             if (isFalling) {
158                 animator.SetBool("IsClimbing", true);
159             }
160             else {
161                 animator.SetBool("IsClimbing", false);
162             }
163         }
164         #endregion
```

Obrázek 18 - PlayerController, metoda FixedUpdate, snímek 3/3, region Ladder snímek 2/2

Poslední část kódu řeší samotný pohyb a animace při kolidování se žebříkem. Podmínka **if** ověřuje, zda se **isClimbing** rovná **true** a zároveň paprsek **Raycast** koliduje s nějakým objektem. Jestli je podmínka true, zjistí se ze vertikální vstup z klávesnice, který se vloží do proměnné **moveVertical** a ta se dále předá do vlastnosti **velocity** (rychlosti) v **Rigidbody2D** jako parametr **osy y** vynásobená proměnnou movementSpeed. Do parametru **osy x** se přiřadí současná hodnota rychlosti hráče ve scéně. Pro objekt se vypne gravitace a do komponentu Animator se nastaví hodnota IsClimbing na true. Pokud podmínka vrátí false, vyhodnotí se else, ve kterém se gravitace zapne a booleanu isClimbing se přiřadí hodnota false. Poté se vyhodnocuje vnořená podmínka if, která kontroluje pravdivost proměnné isFalling. Jestliže vrátí true, **setter** nastaví hodnotu IsClimbing na true a naopak.

5 NÁVRH A TVORBA GUI

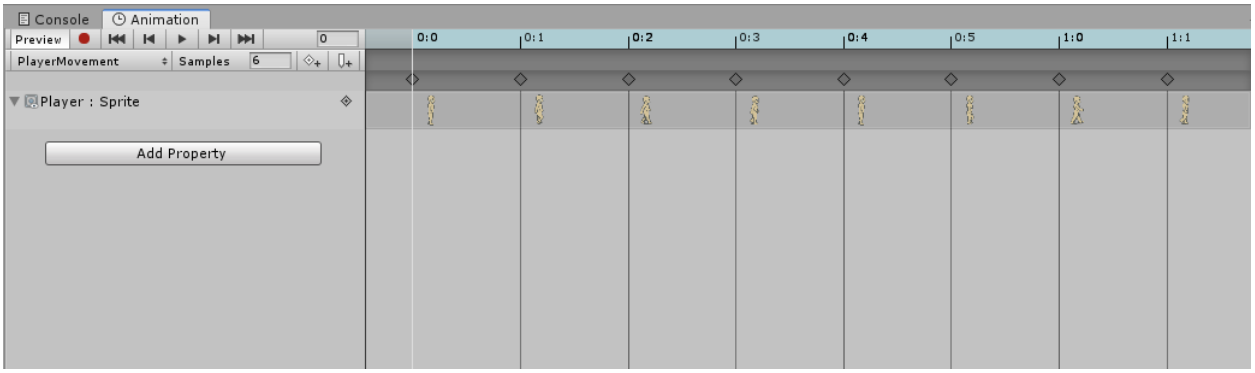
Grafiku UI jsem tvořil v programu Piskel. Tedy vše, co se nachází na Canvasu scény. Pracoval jsem s Assetem z Asset Storu Fantasy Wooden GUI: Free.[6] Tímto Assetem jsem se buď nechával inspirovat a tvořil sám vlastní obrázky, nebo jsem vzal obrázek přímo z Assetu a ten zjednodušil či upravil podle potřeb.

Obrázek 19 - Piskel, ukázka tlačítka

Nejvíce jsem používal nástroj pero a zrcadlené pero, které může zrcadlit vertikální a horizontální osu, či obě dohromady. Dále jsem používal nástroj tahu, díky čemuž jsem mohl jednoduše dělat rovné úsečky. Samozřejmě jsem využíval i nástroje kbelík, guma a kapátko. Nejčastěji jsem pracoval s paletou barev na obrázku, tudíž odstíny hnědé.

Pozadí scény Home jsem našel na stránce Reddit. [7]

Grafiku hlavní postavy jsem stáhl z internetu, kde je zdarma ke stažení a používání. [8]
Z originálního obrázku jsem si každou část pohybu rozdělil do samostatných souborů, abych je posléze mohl naanimovat v Unity editoru.



Obrázek 20 - Animace postavy

Grafiku domu a konzumovatelných předmětů pro mě zadarmo vytvořil spolužák Patrik Schneider ve Photoshopu CS6.

6 VÝVOJ PROGRAMU

Hráč mu svůj StaminaBar – ukazatel výdrže. Potřeboval jsem, aby se ukazatel pohyboval s hráčem a reálně mu ukazoval, kolik má aktuálně výdrže a zda se mu doplňuje. Proto jsem objekt StaminaBar přiřadil k Playerovi jako potomka, to zajistilo pohyb ve scéně. Pro druhý problém jsem vytvořil skript.

```
5   public class StaminaBar : MonoBehaviour {
6
7       // Vektor
8       Vector3 localScaleV;
9
10      void Start () {
11          // Načtení rozměrů objektu
12          localScaleV = transform.localScale;
13      }
14
15      void Update () {
16          // Do rozměru X se načte hodnota pro aktuální proměnnou 'stamina' v %
17          localScaleV.x = PlayerController.stamina / 100.0f;
18          transform.localScale = localScaleV;
19
20          // Skrytí 'staminaBar', pokud je proměnná 'stamina' na 100%
21          if (localScaleV.x >= 1.0f) {
22              gameObject.GetComponent<Renderer>().enabled = false;
23          }
24          else {
25              gameObject.GetComponent<Renderer>().enabled = true;
26          }
27      }
28  }
```

Obrázek 21 – skript StaminaBar

Definoval jsem si proměnou **localScaleV**, typu Vector3 UnityEnginu**,** pro osy **x, y, z**. V metodě Start do této proměnné vloží reálná velikost ve scéně zohledňující škálování velikosti rodiče. Potom se v metodě Update do proměnné **localScaleV** v ose x přiřadí aktuální hodnota výdrže ze skriptu PlayerController vydělená 100. Přidělí se desetinné číslo, ale můžeme tuto hodnotu chápat jako procenta. Ta se odešle zpět do komponentu **Transform**. Tento kousek kódu zajišťuje grafický přehled výdrže hráče. Dále se prochází podmínka **if**, kde se vyhodnocuje, zda proměnná není větší nebo rovna 1. Znovu si toto číslo můžeme představit jako 100 %. Pokud podmínka vrátí **true**, vypne se vykreslování objektu na scéně, aby nám ukazatel nezavazel, když se s ním zrovna nebude nic dít. Jinak se nám stav ukazatele výdrže zobrazuje pořád.

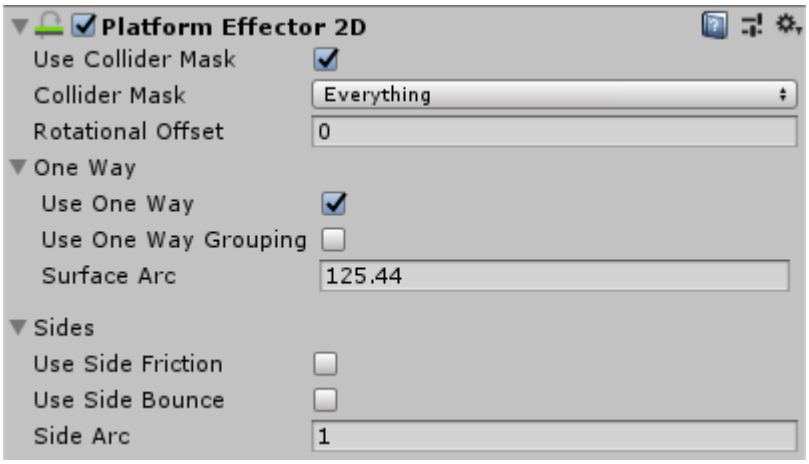
Další problém nastal při konzumaci potravin, kdy hodnoty nabývaly záporných hodnot, pokud byla hodnota, například hladu, menší než číslo, kterým se odčítala. Proto jsem navrhl **For cyklus** s vnořenou podmínkou **if**, aby se hodnota zmenšovala postupně a každý krok se kontroloval, aby se nedostala do záporných hodnot.

```
51          if (isPressed == true && (hitInfo.collider != null || hitInfoRightPickUp.collider != null
52              || hitInfoLeftPickUp.collider != null)) {
53              Destroy(gameObject); // Zničení objektu
54              player.HygieneIncrement();
55
56              // For cyklus pro snížení hodnoty v proměnné 'hunger' ve scriptu PlayerController
57              for (int i = 0; i < 20; i++) {
58                  if (PlayerController.hunger < PlayerController.maxHunger && PlayerController.hunger > 0)
59                      PlayerController.hunger--;
60              }
61          }
```

Obrázek 22 - skript FoodController, konzumace

Konzumaci předchází kontrolovací podmínka **if**, která kontroluje pravdivost booleanu **isPressed** a kolizi paprsků **Raycast**. Pokud podmínka vrátí **true**, objekt vázaný na skript FoodController se zničí a zavolá se metoda **HygieneIncrement** třídy PlayerController, která ve třídě přičte do proměnné hygiene číslo 20. Poté následuje již zmíněný **For cyklus**, který postupně v proměnné snižuje hodnotu o 1. Zároveň se kontroluje, zda je proměnná v reálném rozmezí.

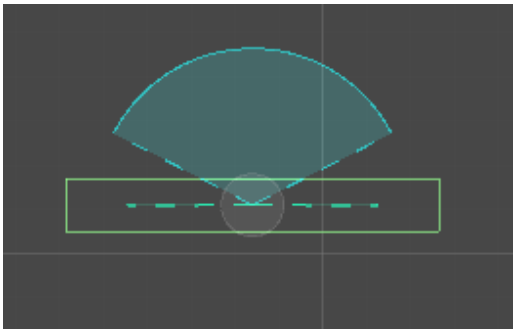
Další překážkou bylo slézání po žebříku, aby hráč slezl jen když opravdu chce a nepropadával, když jde pouze nad ním. Použil jsem komponent Unity **Platform Effector 2D**, který se váže na komponent Box Collider 2D.



Obrázek 23 - Platform Effector 2D

Zde jsem pracoval s funkci **One Way**, která způsobovala to, že objekt snímal kolizi jen z jedné strany. Úhel snímání se potom dá upřesnit v okénku **Surface Arc**.

Ten jsem upravil tak, aby musel hráč stát nad žebříkem a nenastával problém, kdy se hráč sekal o okolní podlahu.



Obrázek 24 - Platform Effector 2D, Surface Arc

Dále jsem přes skript řešil procházení a úpravu platformy, aby se chovala dynamicky.

```
5   public class Platform : MonoBehaviour {
6
7       // PUBLIC proměnné
8       public float waitTime; // časovač
9
10      // PRIVATE komponenty/objekty
11      private PlatformEffector2D effector;
12
13      void Start () {
14          // Načtení komponentů objektu
15          effector = GetComponent<PlatformEffector2D>();
16      }
17
18      void Update () {
19          // Časovač pro projití platformy
20          if (Input.GetKeyUp(KeyCode.S)) {
21              waitTime = 0.5f;
22          }
23
24          if (Input.GetKey(KeyCode.S)) {
25              if (waitTime <= 0) { // Časovač na 0
26                  effector.rotationalOffset = 180f; // Otočení snímání kolize platformy o 180°
27                  waitTime = 0.5f; // Resetování časovače
28              }
29              else {
30                  waitTime -= Time.deltaTime; // Odpočítávání z časovače
31              }
32          }
33
34          if (Input.GetKey(KeyCode.W)) {
35              effector.rotationalOffset = 0f; // Otočení snímání kolize platformy o 0° (defaultní)
36          }
37      }
38  }
```

Obrázek 25 - skript Platform

Vytvořil jsem proměnnou **waitTime**, která mi poslouží jako časovač, či tzv. delay. Referenci na komponent v objektu pod názvem **effector**. Klasicky si v metodě Start vložíme komponent do proměnné, abychom s ním mohli pracovat. Následuje metoda Update s několika podmínkami **if**.

V první podmínce se ze vstupu zjišťuje, zda byla klávesa S puštěna. V případě návratové hodnoty **true** se časovač **waitTime** nastaví na hodnotu 0,5.

V druhé podmínce se kontroluje opět klávesa S ze vstupu, avšak pokud je stisknuta. Následuje vnořená podmínka, která vrací **true**, pokud je časovač **waitTime** menší nebo roven nule. Poté se provede otočení snímání kolize platformy o 180° skrz referenci na funkci komponentu, tudíž místo nad ním, bude snímat kolizi pod ním. V tu chvíli hráč projde a časovač se tedy může resetovat. V případě false se provede else, kde se hodnota v proměnné waitTime postupně odčítá.

Třetí podmínka řeší událost, kdy hráč přistupuje k platformě zespodu pod žebříku, takže drží tlačítko W, snímaní kolize platformy se přetočení zpět na 0°.

7 TESTOVÁNÍ A ODLADĚNÍ

Testování programu a zkoušení všech možných variant, které mohou nastat a tudíž nastanou, je základ.

7.1 Bugy

Na první bug jsem narazil při testování hráčovy animace pohybu s přetáčením spritu. Animace mi fungovala správně, když šel hráč doprava. Problém nastal při přetočení a chůze doleva, kdy animace přestala fungovat. Problém byl v tom, že v Animatoru se přehrávala chůze, pokud byla hodnota Speed větší než 0 a animace stání, když byla hodnota menší než 0. Jenže **Input.GetAxisRaw** vrací hodnoty -1, 0, 1. V Animatoru není možnost porovnání, zda je hodnota rovna číslu, proto jsem to vyřešil skrz matematickou funkci **Mathf.Abs** a odesílal jen kladnou hodnotu ze vstupu.

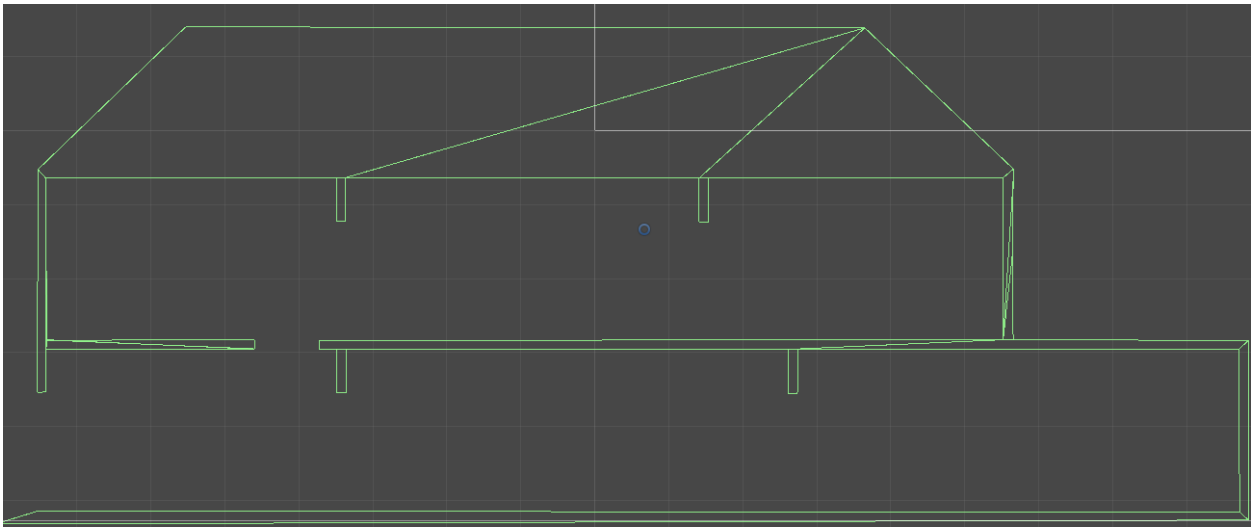
Druhý problém mi nastal při řešení konzumace jídla po stisknutí klávesy. Mým cílem byl objekt v pozadí, skrz který se dá procházet a reaguje pouze na zmiňovaný vstup z klávesnice. První verze obsahovala komponent Box Collider 2D, přes který jsem poté řešil Unity metodu **OnTriggerEnter2D**, která se vypořádává s kolizí s jinými objekty. Jenže se přes objekt nedalo projít a nastavení funkce **isTrigger** na komponentu sice povolila procházení, ale deaktivovala metodu OnTriggerEnter2D, tudíž se přestala řešit kolize a nemohlo ke konzumaci dojít. Další verze obsahovaly pouze komponent Rigidbody 2D a s ním spjatou metodu **OnCollisionEnter2D**, avšak tyto pokusy taktéž neuspěly. Nakonec jsem problém vyřešil přes paprsky **Raycast** z funkcí fyziky Unity. Ty kontrolovaly vektory do stran, zda narazily na hráče a odesílaly signál k zobrazení možnosti konzumace.

Nastalo i pár bugů, které zapříčiňovaly chybějící importy, které jsem si posléze našel v dokumentaci Unity.

Další odladění nejsou ani tak bugy, ale spíš efektivita a logika práce, aby bylo řešení co nejefektivnější.

7.2 Odladění

Sprite domu je ve scéně jako jeden objekt a přiřazení X Box Colliderů by bylo neefektivní a programátorky nekorektní. Proto jsem si v dokumentaci našel Polygon Collider 2D u kterého je možné si vyformovat tvar dle potřeby. Můžete si již v komponentu určit počet vertexů ze kterých budete vytahovat polygony, či si je tvořit editování.
Ukázka Collideru na další stránce.



Obrázek 26 - Polygon Collider 2D

Pro další usnadnění práce jsem musel vyzrát u přidávání konzumovatelných objektů do scény. Jelikož atributy a metody objektů budou stejné a pro škálovatelnost je zapotřebí určitá work flow, bylo třeba zakomponovat dědění. To Unity řeší velice dobře pomocí Prefabů. Ty v sobě dokáží ukládat objekt, jeho komponenty a všechny potomky. Ty se poté mohou znovu používat a přenášet do scény podle potřeby. V Hierarchii je můžete najít pod modrou ikonkou. Objekty poté můžete upravovat buď ve scéně nebo v souboru s Prefabami. Pokud upravíte objekt jednoho potomka Prefabu ve scéně, můžete změny aplikovat na všechny ostatní, nebo jen na něj. Pokud si to rozmyslíte, můžete změny reverzovat, čili vrátit zpět.

8 POUŽITÁ LITERATURA A ZDROJE

[1] Asset Store. *Unity* [online]. Amerika: Unity Technologies, 2010 [cit. 2019-03-25]. Dostupné z: https://assetstore.unity.com

[2] Unity Documentation. *Unity* [online]. Amerika: Unity Technologies, 2005 [cit. 2019-03-25]. Dostupné z: https://docs.unity3d.com/Manual/

[3] Unity Store. *Unity* [online]. Amerika: Unity Technologies, 2005 [cit. 2019-03-25]. Dostupné z: https://store.unity.com/products/unity-personal

[4] Photoshop. *Photoshop* [online]. Amerika: Adobe Systems, 2003 [cit. 2019-03-25]. Dostupné z: https://www.photoshop.com

[5] Visual Studio. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-15]. Dostupné z: https://cs.wikipedia.org/wiki/Microsoft_Visual_Studio

[6] HAMMER, Black. Asset. *Unity* [online]. Amerika: Unity, 2017 [cit. 2019-03-18]. Dostupné z: https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811

[7] Background. *Reddit* [online]. Amerika: Reddit, 2018 [cit. 2019-03-18]. Dostupné z: https://www.reddit.com/r/PixelArt/comments/8fdfbv/newbie_made_a_pixel_art_city_first_real_project/

[8] STEELE, Brett. Avatar. *OpenGameArt* [online]. Amerika: OpenGameArt, 2011 [cit. 2019-03-18]. Dostupné z: https://opengameart.org/content/2d-sprite-skins-walking-animation

[9] Stack Overflow. *Stack Overflow* [online]. Amerika: Stack Exchange, 2008 [cit. 2019-03-25]. Dostupné z: https://stackoverflow.com

9 RESUMÉ (V ČESKÉM A ANGLICKÉM JAZYCE)

9.1 Česky

V semináři programování jsem vyvíjel hru v herním enginu Unity. Programovacím jazykem byl C#. Hra je na způsob simulátoru běžného života. Využil jsem objektově orientované programování za cílem škálovatelnosti hry.

9.2 English

I was developing a game with a game engine called Unity in my programming seminar throughout the year. The programming language was C#. The game is based on a simulator of a daily life. I used object oriented programming to ensure game scalability.

10 ZÁVĚR

Za půl roku vývoje programu jsem se naučil mnoho nového, prohloubil své programovací znalosti a získal úplně jiný náhled na objektově orientované programování. Narazil jsem na hodně překážek, ale díky dokumentaci a video tutoriálech Unity jsem dokázal většinu věcí přelousknout a v ostatních případech tu pro mě byl Stack Overflow.[9] S výsledkem práce jsem spokojen i když jsem si na začátku určil až moc velké cíle, které nebylo možné splnit, jako asi každý začátek v tvorbě videoher. Ovšem základní cíle byly vytvořit simulátor běžného života, vytvořit si konzumovatelné objekty, jeden atribut bude ovlivňovat druhý, přenášení hodnot proměnných mezí scénami a podobné. A to jsem splnil tak, že z dlouhodobého hlediska by se s hrou dalo pracovat a to pro mě bylo důležité.

Hru najdete na DVD v souboru Píšek. Ten otevřete a spustíte exe soubor Pokus.