STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

2D logická hra v Unity

Martin Vrba
Liberecký kraj

Liberec 14.03.2022

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

2D logická hra v Unity
2D puzzle game in Unity

Autoři: Martin Vrba
Škola: Střední průmyslová škola strojní a elektrotechnická a Vyšší odborná škola, Masarykova 3, 461 Liberec
Kraj: Liberecký kraj
Konzultant: Mgr. Michal Stehlík

V Liberci dne 14.03.2022

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Liberci dne 14.03.2022 ……
Martin Vrba

Anotace

Práce se zabývá vývojem logické hry založené na japonské hře Sokoban. Popisuje vnitřní fungování samotné hry, vývoj vlastního nástroje pro vlastní úrovně a možné využití v budoucnu.

Annotation

This work deals with development of a puzzle game based on Japanese videogame Sokoban. It describes internal functions of the game itself, custom tool for generating own levels, and possible future use and expansion.

Obsah

Úvod

Programování bylo mým koníčkem už od třetí třídy základní školy, kdy jsem se poprvé setkal s roboty Lego Mindstorms, na kterých jsem pochopil koncept proměnných, smyček a podmínek. Od té doby jsem se sám učil základům jazyků jako Javascript, PHP, nebo Python. Sice jsem za tu dobu vytvořil mnoho miniaturních projektů, většina ale sloužila pouze jako experiment pro získání více zkušeností. Teprve když jsem začal studovat na průmyslové škole v Liberci, naučil jsem se dobrým návykům, organizaci kódu a obecně konceptu objektově orientovaného programování. Díky tomu jsem byl schopen uplatnit své znalosti i ve vývoji her v enginu Unity, který mi do té doby nedával tolik smysl. Cílem této práce je vytvořit hru, která bude jednoduše rozšiřitelná o nové mechaniky a úrovně. Nejedná se tedy o finální verzi hry, ale spíš základ, který má úrovně uložené v externích souborech.

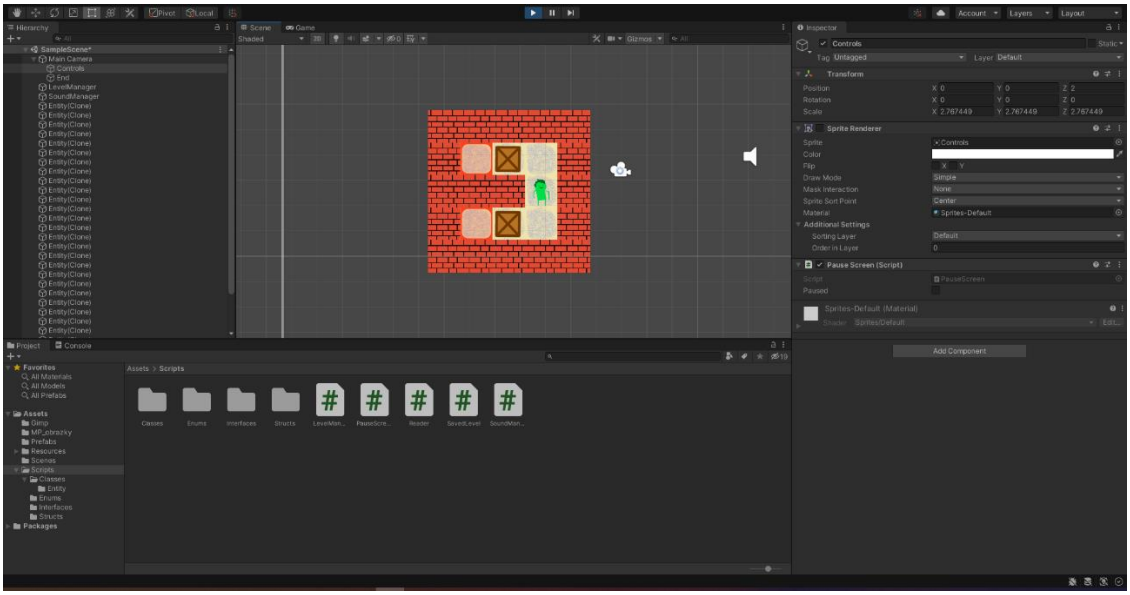
1 POUŽITÉ TECHNOLOGIE A NÁSTROJE

V této kapitole jsou popsány všechny programy a jazyky, které jsem k vytvoření této práce použil.

1.1 Herní engine

Jedno z nejdůležitějších rozhodnutí pro vývoj této hry. Pro mě byl engine Unity lákavým kandidátem už od první chvile, kdy jsem se začal učit objektově orientované programování a jazyk C#. V minulosti jsem si trochu vyzkoušel práci i v Unreal Engine, Godot a Phaser. Myslím si, že jsem se rozhodl správně pro Unity, protože jsem v něm měl již více zkušeností a jazyk C# je ideální pro organizování rozšiřitelného kódu.

1.1.1 Rozhraní Unity



Obrázek 1 Unity3D

Unity je 3D herní engine poprvé vydaný v roce 2005 firmou Unity Technologies. Od té doby byl až do dnes neustále vyvíjen a za ten čas v něm bylo vytvořeno mnoho známých titulů v herním průmyslu jako například:

- Monument Valley
- Ori and the Blind Forest
- Pokémon Go
- The Long Dark
- Inside

- Gone Home
- Furi

Přestože je Unity ve své podstatě engine pro trojrozměrné hry, má velmi dobrou podporu i pro 2D grafiku a mechaniky.

Unity pracuje s takzvanými herními objekty (GameObject). Každý má svoje vlastnosti (například název, pozici v prostoru, rotaci nebo zvětšení) a komponenty (zvuky, definice hran pro kolize, simulaci fyzikálního tělesa)

1.2 Kód

Vzhledem k tomu, že je má hra postavena hlavně na funkčnosti herních mechanik, je kód velmi důležitou součástí vývoje a budoucího rozšiřování.

1.2.1 Programovací jazyky

Základní jazyk v Unity je C# (C Sharp). Je objektově orientovaný a syntaxí je podobný jazykům jako Java nebo C++. C# se běžně využívá pro vytváření desktopových, mobilních, i webových aplikací. Unity má svojí vlastní knihovnu pro ovládání prvků jménem UnityEngine, která obsahuje mnoho užitečných funkcí a metod. Dá se pomocí ní dotazovat na různé podmínky běžící hry jako například velikost okna, pozice myši, čas mezi vykreslenými snímky hry a mnoho dalších informací, které budou ještě zmíněny. V jazyce C# je psaná veškerá logika hry a všechny skripty pracující se soubory a herními objekty.

Přestože všechen kód v Unity se programuje v jazyce C#, zahrnuje moje práce i webový editor úrovní, který byl vytvořen v kombinaci HTML, CSS a Javascript. Je to statická stránka, kterou lze otevřít offline ve webovém prohlížeči. Na předávání informací o levelu se používá datový formát JSON (JavaScript Object Notation), takže lze úroveň převést z Javascriptu do C# bez problémů s kompatibilitou. Konkrétní provedení aplikace bude ještě podrobněji vysvětleno.

1.2.2 Vývojové prostředí

1.2.2.1 Visual Studio

Visual Studio je IDE (vývojové prostředí) od firmy Microsoft, určené k vyvíjení programů v jazycích jako C#, C++, Python a dalších. Pro mé účely funguje jako editor C# skriptů v Unity. Má automatické doplňování klíčových slov a přehledné uživatelské rozhraní pro opravu chyb v syntaxi. Dělá mnoho pracných činností za programátora, například při používání externích metod importuje potřebné knihovny pomocí jednoho kliknutí myši.

1.2.2.2 VS Code

Visual Studio Code je opensource editor kódu od firmy Microsoft. Na rozdíl od klasického Visual Studia zabírá VS Code o dost méně místa na disku a zároveň ho lze spustit i na macOS a Linuxu. Pro mě je tento editor příjemnější pro vývoj v HTML a Javascriptu. Vytvořil jsem v něm tedy celý editor levelů.

1.3 Audio

K vytvoření veškerého audia jsem použil programy sfxr, FL Studio a Audacity, každý k jinému účelu.

1.3.1 Sfxr

Sfxr je malý opensource program vytvořený Tomasem Petterssonem, který umožňuje generování 8-bitových zvukových efektů. Lze v něm upravovat a mutovat parametry, takže je možné se postupně dopracovat k žádoucímu zvuku. V sfxr byly vytvořeny zvukové efekty.

1.3.2 FL Studio

Původně zvané FruityLoops, FL Studio je DAW (digitální zvukové studio) vyvíjené belgickou společností Image-Line. Používá se převážně na vytváření hudby. Má čtyři edice lišící se v ceně, některých vlastnostech a funkcích. Existuje sice bezplatná zkušební verze, ale já jsem měl program zakoupený, takže jsem toho využil. V FL Studiu byla vytvořena hudba na pozadí.

1.3.3 Audacity

Audacity je známý opensource audio editor. Umožňuje aplikovat velkou škálu efektů na audio klipy a zároveň je mixovat či stříhat. Obvykle využívám Audacity k více účelům, ale zde jsem pouze upravoval hlasitosti jednotlivých zvuků a odstraňoval šum.

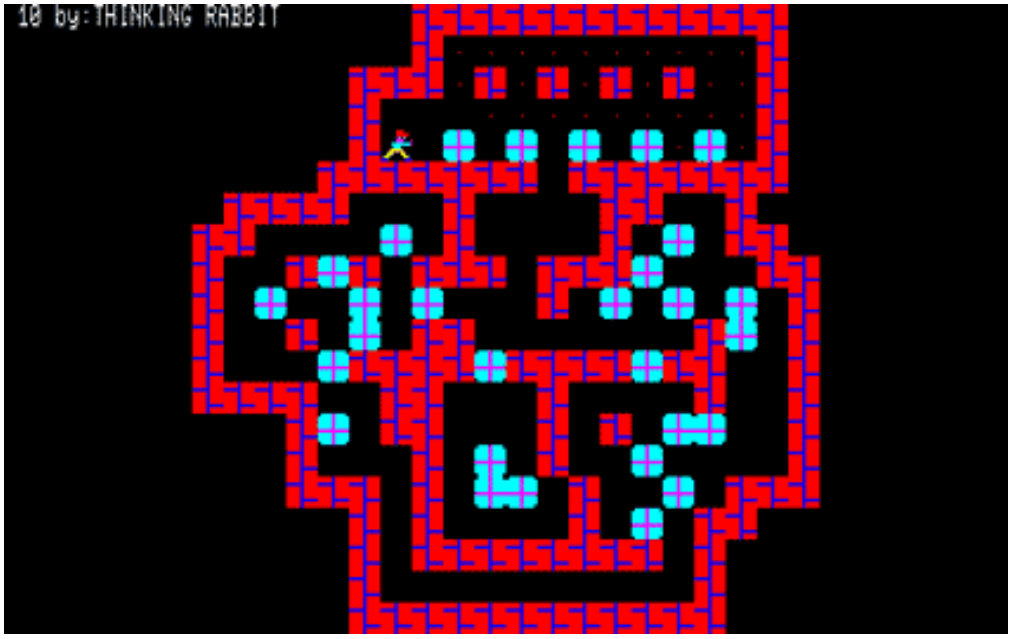
1.4 Grafika

Všechny assety byly vytvořeny v programu GIMP (GNU Image Manipulation Program). Je to open source editor obrázků, který byl původně vytvořen jako Linux alternativa pro Adobe Photoshop.

2 PRINCIP HRY

2.1 Sokoban

Základní mechaniky hry jsou založené na starší japonské hře Sokoban (česky Skladník). V její nejjednodušší formě se jedná o 2D hru v mřížce, kde hráč pohybuje jednou postavou. Jeho úkolem je dostat všechny bedny na předem určená místa. Bedny může pouze tlačit před sebou, a to maximálně jednu. Nikdy nejde tlačit více beden, nebo naopak bedny tahat za sebou.



Obrázek 2 Originální Sokoban (1984)

2.2 Vlastní rozšíření

Původní plán byl vytvořit hru, která se bude Sokobanem pouze inspirovat, ale bude mít mnoho prvků navíc, které by zásadně rozšířily možnosti kreativity při tvoření levelů. Mezi takové prvky by mohlo patřit například:

- Led, který omezí volný pohyb. Místo toho bude pokračovat ve stejném směru, dokud se nedostane na druhou stranu. Led by ovlivnil i ostatní prvky.
- Dveře s propojenými tlačítky a klíči. Dveře blokují hráči a objektům průchod, dokud nejsou odemčeny buď stáním na tlačítku, nebo přímým odemčením pomocí klíče. Samozřejmě by byly možné i další způsoby, jak některé z dveří ovládat.
- Skleněné bedny a případně jiné objekty, které blokují tlačení beden, a hráč je musí rozbít přímým dotykem.

- Nestabilní podlaha. Když se jakýkoliv objekt ocitne na tomto poli, aktivuje nestabilní podlahu. V momentě, kdy všechny objekty opustí aktivní nestabilní podlahu, vytvoří se na původním místě díra, která blokuje průchod hráče. Bylo by možné přidat ještě další mechaniku, kde přebytečné bedny lze zatlačit na pole díry, a tím vytvořit jakýsi most. Do té doby by nestabilní podlaha fungovala jako jednosměrná překážka.
- Robot chodící ze strany na stranu, který tlačí vše před sebou. Udělá krok po každém kroku hráče, ale i když hráč počká na místě. Robot by mohl na rozdíl od hráče tlačit více beden najednou. Mezi různá řešení úrovní by mohlo patřit i načasování tahů tak, aby robot tlačil bedny ve vhodný okamžik přes led a hráč je pak dostal na místo, kam by to sám nezvládl.
- Unikátní bedny, které musí být umístěny na konkrétní místo a nelze je zaměňovat. Byly by pravděpodobně odlišeny barvou.

Během budoucího rozšiřování hry jistě přijdu na různé úpravy těchto nápadů nebo dokonce vymyslím i nějaké další.

2.3 Cíl vývoje

Cíl, ke kterému jsem se vydal krátce po založení projektu, nebylo udělat konkrétní hru obsahující výše zmíněné prvky a mechaniky. Je tím naopak vytvořit jakýsi engine, do kterého lze vložit vygenerované levely, které Unity automaticky pomocí herních objektů zobrazí jako ovladatelnout hru.

3 VÝVOJ

Celý proces vývoje byl o dost delší, než jsem si původně myslel. Hodně jsem toho musel programovat více než jednou, protože jsem neodhadl, jak komplexní některé metody budou, nebo co všechno chci mít rozšiřitelné v pozdějším vývoji.

3.1 Datová struktura

Původně jsem plánoval mít herní prvky zakódované pomocí čísel v dvourozměrném poli. Toto pole by se po každém tahu aktualizovalo a následně promítlo na obrazovku pomocí pozicovaných obrázků. Velice brzo jsem ale přišel na problém, který se ukázal být až kritickým. V tomto systému by bylo obtížné zakódovat více objektů na jednom místě – v tomto bodě bylo pro mě ještě pořád důležité udělat plnou hru s více komplexními mechanikami, mezi které by mohly patřit i objekty naskládané na sobě. Další problém, se kterým jsem se setkal, byla neschopnost tohoto systému zakódovat některé vlastnosti objektů, jako je například směr. Rozhodl jsem se tedy vymyslet lepší způsob, jak uchovávat informace o herním poli a jeho obsahu. Více efektivním a rozšiřitelným postupem bylo využití objektově orientovaného programování a vytvoření systému dědičnosti a rozhraní.

3.1.1 Vlastní datové typy

Aby se mi v kódu lehce pracovalo s daty, vytvořil jsem si několik pomocných datových typů a naprogramoval i vlastní operátor pro zjednodušení zápisu.

3.1.1.1 TileType

Tento enum je zde hlavně pro konvertování z JSON souboru na samostatné třídy. Obsahuje výčet všech možných entit.

```
public enum TileType
{
 Floor,
 Wall,
 Player,
 Box,
 Storage
}
```

3.1.1.2 Direction

Direction je enum pro značení směru. Kromě běžných čtyř směrů je zde ještě pátá možnost pro žádný směr.

```
public enum Direction
{
    None = 0,
    Up = 1,
    Down = 2,
    Left = 3,
    Right = 4
}
```

3.1.1.3 Coordinates

Pro zapisování souřadnic jsem použil struct o dvou proměnných (pozice X a Y). Mimo to je zde i vlastní operátor pro „přičítání“ směru k souřadnicím. Funguje to tak, že pokud by se k souřadnici měl přičíst směr nahoru, vrátí operátor novou souřadnici,

3.1.2 Entity

Entitou se v mé práci myslí jakákoliv věc, která bude mít svůj vlastní sprite. Samotná třída Entity neměla na začátku vývoje žádné vlastnosti kromě souřadnic a odkazu na GameObject, který k ní byl vytvořen a přiřazen.

```
public abstract class Entity
{
    public GameObject MappedObject;
    public Coordinates Position;
    protected LevelState state;
    protected SpriteRenderer renderer;
    protected Dictionary<string, Sprite> sprites;

    //... metody pro ovladani sprite
```

Z této třídy by se následně vytvářely dědící třídy, každá pro jinou entitu.

3.1.3 Rozhraní

Abych si udržel pořádek v tom, která entita dělá jaké činnosti, vytvořil jsem si rozhraní pro vlastnosti, které mohou jednotlivé entity mít. Mám v plánu v budoucnosti přidat alespoň některé z rozšiřujících mechanik, takže následující list je pouze pro objekty v původním Sokobanovi.

3.1.3.1 IDirectionFacing

Všechny entity, které mají schopnost se otáčet nebo být orientovány konkrétním směrem, mají rozhraní IDirectionFacing. Obsahuje metody pro získání a modifikování směru a zároveň souřadnici, na kterou se entita právě „dívá“. V tuto chvíli je jedinou entitou s tímto rozhraním samotná postava hráče.

```
public interface IDirectionFacingEntity
{
 public Direction GetDirection();
 public void SetDirection(Direction newDir);
 public Coordinates LookingAt();
}
```

3.1.3.2 IMovingEntity

Cokoliv, co se může pohybovat (ať už samo, nebo pomocí jiné entity) má rozhraní IMoving. Obsahuje pouze metodu pro posunutí na nové souřadnice.

```
public interface IMovingEntity
{
 public void Move(Coordinates destination);
}
```

3.1.3.3 IObstacle

IObstacle je pozůstatek z původního nápadu na dveře otevíratelné tlačítky. Obsahuje možnost získat a kontrolovat, zda je entita s IObstacle „otevřená“. Nyní má toto rozhraní pouze Wall a je konstantně zavřená.

```
public interface IObstacle
{
```

```
 public bool Opened { get; }
 public void Open(bool open);
}
```

3.1.3.4 IPushable

Jediná entita, kterou může hráč momentálně posouvat, je Box. Přišlo mi lepší tuto vlastnost zobecnit, kdybych v budoucnosti chtěl přidat jiné posouvatelné prvky.

Metoda Push je typu bool proto, aby mohla dávat zpětnou vazbu, jestli akce posunutí proběhla, nebo entitě něco brání.

```
public interface IPushable
{
    bool Push(Direction direction);
}
```

3.1.4 Levely

Už od začátku jsem věděl, že chci levely mít nějak zakódované do souborů, aby se s nimi dalo jednoduše zacházet a importovat je. Vytvořil jsem si tedy třídu SavedLevel, která je serializovatelná (tj. dá se převést z instance třídy na JSON nebo naopak) a obsahuje pole EntityConstructor, z čehož se dá vytvořit každá entita v levelu.

```
[System.Serializable]
public class SavedLevel
{
 public string Name;
 public EntityConstructor[] Entities;
}
```

EntityConstructor je také serializovatelná třída, obsahující všechny informace o entitě, která bude vytvořena.

```
[System.Serializable]
public class EntityConstructor
{
 //Entity
```

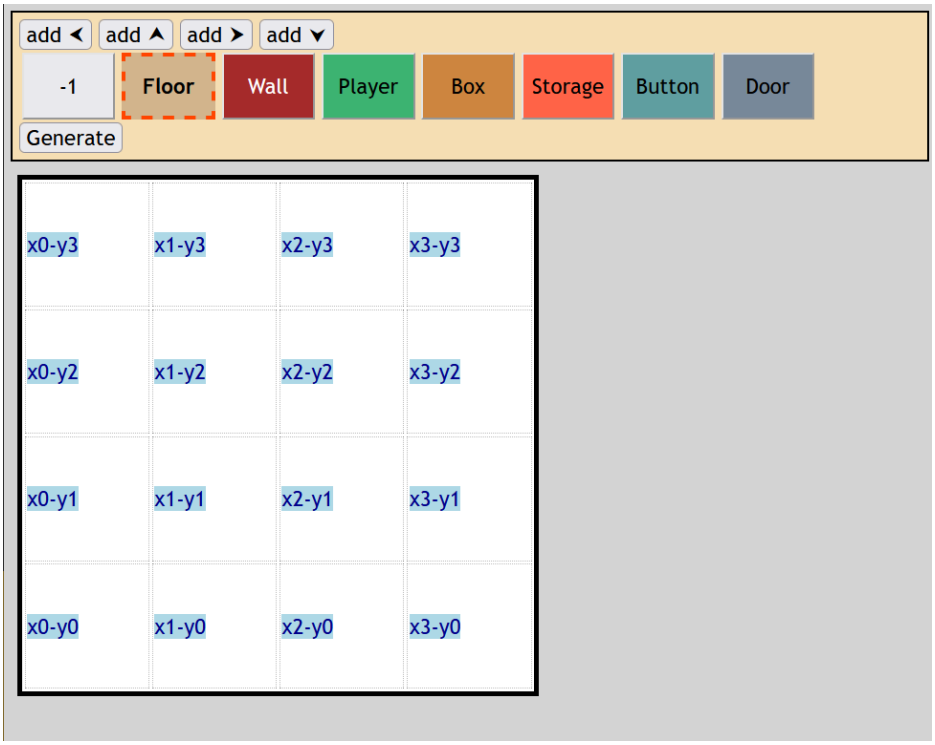
```
 public TileType T; //Type
 public int CoordinateX;
 public int CoordinateY;

 //IDirectionFacing
 public Direction Direction;

 //IObstacle
 public bool Opened;
}
```

Se SavedLevel pracuje asi nejdůležitější třída v celé práci, a to LevelManager, která má na starost všechnu režii hry, vnímá vstupy hráče, načítá nové levely a ovládá všechny entity.

```
public class LevelManager : MonoBehaviour
{
    //vlastnosti a promenne
    // ...

    void Start() {…}

    void ReadAllLevels() {…}
    bool LoadLevel(int id) {…}

    void Update() {…}
    void Undo() {…}
    void Restart() {…}
    void Tick(Direction pressed) {…}
    void Render() {…}
    void RenderTile(Entity entity, int z) {…}
}
```

3.2 Editor

Abych nemusel levely ručně psát do souborů, vytvořil jsem si vlastní editor pomocí HTML a JavaScript. K dokonalosti má sice daleko, ale jako nástroj pro tvoření vlastních úrovní postačí.



Obrázek 3 Rozhraní editoru

Nynější verze editoru umožňuje následující akce:

- zvětšovat plochy mřížky
- vybrat typ entity a následně jej přidat na políčka
- vyplnit souvislou prázdnou plochu podlahou
- vygenerovat JSON soubor

3.3 Cyklus hry

3.3.1 Načtení

Po spuštění programu se do paměti uloží odkazy na všechny JSON soubory v adresáři určeném pro levely.

```
//LevelManager.cs
void ReadAllLevels()
{
```

```
        levelTexts = Resources.LoadAll<TextAsset>(path);
}
```

Následuje samotné načtení první úrovně. LevelManager si pamatuje, kolikátý level je právě načten, takže lze jednoduše postupovat v řadě. Při každém tahu se zkontroluje, zda je již úroveň splněna. Pokud ano, tak se provede načtení následujícího JSON souboru. Pokud již není žádný další level, zobrazí konec hry.

Po úspěšném zvolení souboru pro načtení se daný JSON zparsuje do SavedLevel. Pro každý EntityConstructor se vytvoří nová entita a k ní se automaticky přiřadí nová instance GameObjectu, aby se dala pozicovat a měnit sprite.

```
for (int i = 0; i < savedLevel.Entities.Length; i++)
{
    EntityConstructor constructor = savedLevel.Entities[i];
    GameObject newObject = Instantiate(prefabEntity);
    Entity newEntity = null;
    switch (constructor.T)
    {
        case TileType.Floor:
            newEntity = new Floor(constructor, newObject);
            break;

        case TileType.Wall:
            newEntity = new Wall(constructor, newObject);
            break;

        // ...
    }
    Environment.Add(newObject);
    levelState.Add(newEntity);
}
```

3.3.2 Vykreslování

Když je hotový LevelState se všemi používanými entitami, je potřeba úroveň vykreslit. Na to je v LevelManageru metoda Render, která projde entitami, a upraví podle jejich dat propojené GameObjecty. Některé objekty se musí vykreslit nad jinými, takže se zde používá proměnná Z, která ovlivňuje vzdálenost od kamery.

```
void Render()
{
     int z = 0; //floor level
     foreach (Floor floor in level.Floors)
     {
          RenderTile(floor, z);
     }
     z = -1; //object level
     RenderTile(level.Player, z);

     foreach (Wall wall in level.Walls)
     {
          RenderTile(wall, z);
     }

     foreach (Box box in level.Boxes)
     {
          RenderTile(box, z);
     }

     foreach (Storage storage in level.Storages)
     {
          RenderTile(storage, z);
     }
}
```

RenderTile pouze aktualizuje sprite objektu a nastaví jeho pozici.

```
void RenderTile(Entity entity, int z)
```

```
{
    entity.MappedObject.transform.position = new
        Vector3(entity.Position.x, entity.Position.y, z)
        * tileSize;

    entity.UpdateSprite();
}
```

3.3.3 Reakce na vstup

Metoda Update se v Unity automaticky volá mezi každým vykresleným snímkem. Pomocí třídy Input z knihovny UnityEngine lze zjistit, zda od posledního snímku uživatel stiskl nějakou klávesu a případně kterou. V každém cyklu Update se tedy provede dotaz, zda byl zaznamenán nějaký vstup, který by mohl ovlivnit stav hry. Mezi takové vstupy patří:

- pohyb hráče nějakým směrem
- čekání na místě (pro případné budoucí využití v entitách, které se aktivně samy pohybují)
- vracení tahu
- zobrazení pomocného obrázku s ovládáním (to se ale provádí v jiné části kódu)

Pokud je zadán vstup pro pohyb nebo čekání, LevelManager provede Tick, což je provedení vnitřní logiky všech entit. Třída Entity má totiž virtuální (přepsatelnou) metodu PerformTick, kterou si každá děděná entita implementuje sama.

```
//Entity.cs
public virtual void PerformTick(Direction input)
{
    throw new NotImplementedException();
}
```

```
//Player.cs
public override void PerformTick(Direction input)
 {
      if (input != Direction.None)
```

```
        {
            ///change direction
            SetDirection(input);

            Tile destination = state[LookingAt()];

            if (destination.Opened)
            {
                if (destination.Box != null)
                {
                    Box box = destination.Box;
                    if (box.Push(direction))
                    {
                        Move(Position + direction);
                        sound.Play("push");
                    }
                }
                else
                {
                    Move(Position + direction);
                    sound.Play("step");
                }
            }
        }
        else
        {
            Debug.Log("input is none");
        }
    }
```

Mnoho entit ve své implementaci PerformTick používá třídu Tile, což je jakýsi zprostředkovatel mezi entitami a třídou LevelState. Tile obsahuje všechny potřebné informace o daném poli.

```
public class Tile
```

```
{
 private LevelState state;

 public Tile(LevelState state)
 {
      this.state = state;
      Entities = new List<Entity>();
 }


 public IList<Entity> Entities { get; private set; }

 public void Add(Entity entity)
 {
      Entities.Add(entity);
 }

 public void Update(Entity entity)
 {
      state[entity.Position].Add(entity);
      Entities.Remove(entity);
 }

 public bool Opened
 {
      get
      {
           bool hasClosedObstacle = false;
           bool hasFloor = false;
           foreach (Entity entity in Entities)
           {
                if (entity is Floor)
                     hasFloor = true;
                if (entity is IObstacle)
                {
```

```
                        if (!(entity as IObstacle).Opened)
                        {
                              hasClosedObstacle = true;
                        }
                  }
            }
            return hasFloor && !hasClosedObstacle;
      }
 }


 public Box Box
 {
      get
      {
            foreach (Entity entity in Entities)
            {
                  if (entity is Box)
                  {
                        return entity as Box;
                  }
            }
            return null;
      }
 }

}
```

4 MOMENTÁLNÍ STAV HRY

Nyní se hra skládá z těchto prvků:

- Podlaha – pouze značí, kde se hráč může pohybovat.

- Zeď – brání hráči v pohybu.

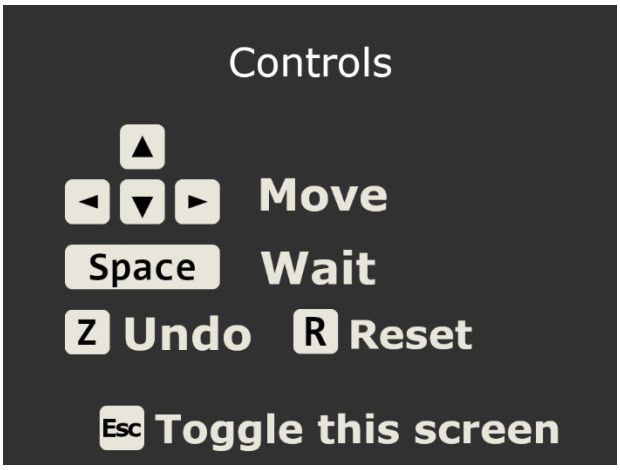
- Hráč – postava, kterou uživatel ovládá.

- Bedna – objekt, který hráč musí dotlačit do skladiště.

- Skladiště – předem vybrané místo pro bedny.

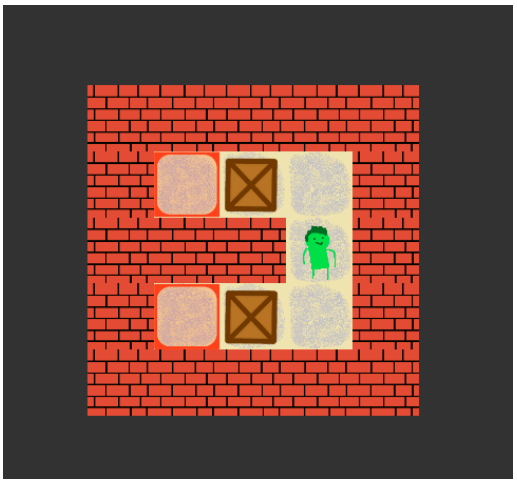
Tyto obrázky jsou dočasné. Jedna z věcí, kterou mám v plánu později dodělat, je kompletní vylepšení grafiky.

Hned po spuštění se zobrazí obrázek s ovládáním.



Obrázek 4 Ovládání

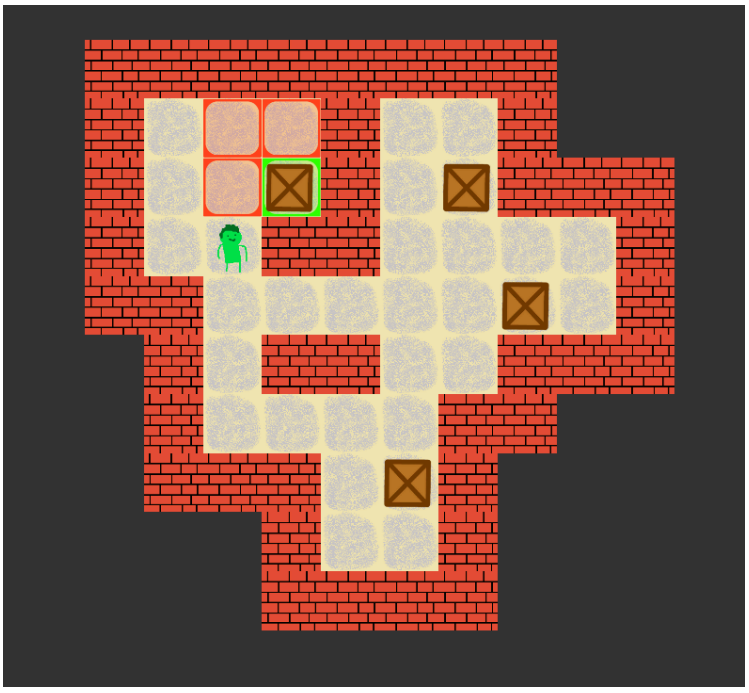
Ve hře jsou zatím jen 4 levely. Jsou zde spíše jako demonstrace mechanik a herního cyklu. První level je moje představa tutoriálu. Velmi jednoduchá úroveň, na které hráč pochopí základ nějakého objektu, nebo určitou novou informaci. Až bude ve hře více objektů, každý bude mít samostatný tutoriál v podobě snadné hříčky pro vysvětlení konceptu.



Obrázek 5 První úroveň

Hráč se může posouvat pomocí šipek do stran nebo čekat na místě. To bude využito, až budou ve hře obsaženy objekty nezávislé na hráči. Ten také může vracet neomezený počet tahů pomocí klávesy Z. Vrátit level do výchozího stavu lze stisknutím klávesy R. Je možné, že se některé ovládání změní.

Cílem hráče je zajistit, aby byly všechny bedny na předem určených místech. Správně umístěná bedna rozsvítí políčko zelenou barvou, takže lze na první pohled vidět pokrok v úrovni.



Obrázek 6 Správně umístěna bedna

Po každém dokončeném levelu zazní zvuk pro indikaci správného řešení. Poté se načte následující úroveň. Pokud hráč projde všemi levely, zobrazí se obrázek značící konec hry.

Obrázek 7 Konec hry

Závěr

Když mě poprvé napadla myšlenka tohoto tématu odborné práce, netušil jsem, kolik se toho vlastně musí vytvořit a zvládnout, než se spustí samotný vývoj obsahu hry. Musel jsem si vytvořit vlastní systém pro ukládání dat, interakce mezi jednotlivými prvky hry, a najít způsob, jak je vykreslovat při každé změně. Určitě by to šlo vyřešit jednodušeji, ale jak už bylo několikrát řečeno, to nakonec nebyl můj cíl. Účelem bylo vytvořit solidní základ, do kterého bude možné přidávat všelijaké nové nápady a mechaniky.

Kdyby výsledkem mé práce byla nerozšiřitelná kopie hry Sokoban, nemělo by to pro mě žádný smysl. Takto jsem naopak získal mnoho zkušeností s organizací práce a mohl přemýšlet nad budoucím vývojem. Hlavní pozitivní stránka mého finálního cíle je to, že mám stále motivaci na této práci dále pokračovat. Snad se mi i podaří implementovat některé z uvedených plánovaných rozšíření. To je něco, co by s původním přístupem nebylo vůbec možné.

Zjistil jsem také, jak důležité je časové rozložení práce. Na začátku jsem měl pět měsíců rozdělených na následující přibližně stejně dlouhé úseky:

1. Datové struktury
    a. Entity
    b. Levely
    c. Pomocné třídy pro parsování z JSON
2. Základní mechaniky
    a. Pohyb hráče
    b. Posouvání beden
3. Levely
    a. Level editor
    b. Zpracovávání vstupu
4. Ostatní
    a. Implementování rozšiřujících entit
    b. Playtestování
    c. Ladění chyb

Toto rozložení mělo zásadní vadu. Ladění kódu většinou zabere více jak polovinu času. Během vývoje jsem narážel na různé problémy s mojí dosavadní strukturou, takže jsem musel vytvářet velké části kódu znovu od začátku.

Vytváření základního systému tříd mi reálně zabralo asi měsíc a půl. Level editor a základní mechaniky jsem dělal najednou. Dohromady jsem na nich pracoval přibližně měsíc. Poměrně náročné bylo propojení herního kódu a externích souborů . Musel jsem kvůli tomu změnit fungování LevelManageru, nejdůležitější třídy. Nestihl jsem ani playtestovat, ani vytvářet nové objekty. Většinu času zabralo opravování kódu. Jsem ale rád, že jsem tomu ten čas dal. Nyní je

datový systém lépe rozšiřitelný a bude se mi mnohem lépe později přidávat a upravovat další obsah.

Základ hry mám sice již hotový, ale do mé představy dokončené hry je stále ještě daleko. Zde jsou některé ze změn, na kterých mám v plánu ještě pracovat:

- Lepší editor levelů. Momentální jednoduchou verzi jsem si vytvořil pouze pro svoje účely během několika hodin. Po každém přidání mechaniky do hry se musí editor ručně aktualizovat. Další důležitá vlastnost, která zde chybí, je editování již existujících souborů. Zatím lze levely pouze generovat z prázdné plochy. Pravděpodobně přepíšu celý backend editoru do C#, aby se vytvořila kompatibilita s daty ve hře.
- Rozšíření hry. Jeden z hlavních důvodů, proč jsem vytvořil vlastní třídy a rozhraní pro entity, je rozšiřitelnost mechanik. S tímto systémem lze přidávat prvky tak, aby zapadaly do již existujícího a fungujícího mechanismu.
- Možnost importování souborů po kompilaci, případně i za běhu. Nyní lze levely importovat pouze do nezkompilovaného projektu Unity. Po zkompilování se soubory zakódují a nelze je jednoduše měnit. Nový systém by tedy měl umožnit otevírat a načítat soubory s úrovněmi rovnou při spuštění nebo později i za běhu. Viděl jsem podobné mechaniky u jiných her vytvořených v Unity, jen jsem se k vlastní implementaci ještě nedostal.
- Podpora vlastních barevných variací a obrázků pro jednotlivé entity, případně i jiných aspektů, jako například vlastní zvuky a hudba. Bylo by to v souladu s mým cílem vytvořit univerzální Sokoban engine, ne jen jednu izolovanou hru.
- Správné pozicování kamery, vypočítané na střed, nebo dynamicky podle velikosti levelu.

Rozhodně nechci na tomto projektu přestat pracovat. Mám pro něj plán a vizi, kterou bych rád naplnil. Je to zatím nejrozsáhlejší programátorská práce, jakou jsem kdy udělal, a nerad bych zahodil příležitost dotáhnout jí do úplného konce.

5 POUŽITÁ LITERATURA

1. **Pettersson, Tomas.** sfxr. *DrPetter's homepage.* [Online] https://www.drpetter.se/project_sfxr.html.

2. **Image-Line.** FL Studio. [Online] https://www.image-line.com/fl-studio/.

3. **Unity Technologies.** *Unity.* [Online] https://unity.com/our-company.

4. **Domènech, Jordi.** Sokoban 30th anniversary. [Online] http://sokoban-jd.blogspot.com/2012/10/sokoban-30th-anniversary-1982-2012.html.

5. **Microsoft.** Visual Studio Code. [Online] https://code.visualstudio.com/docs.

6. **The GIMP Team.** Gimp. [Online] https://www.gimp.org/.

7. **Crockford, Douglas.** JSON. [Online] https://www.json.org/json-en.html.

8. **Audacity.** Audacity. [Online] https://www.audacityteam.org/.

9. **Huber, Thomas Claudius.** [Online] https://www.thomasclaudiushuber.com/2021/02/25/c-9-0-pattern-matching-in-switch-expressions/.

10. **Wikipedia.** Sokoban. [Online] https://en.wikipedia.org/wiki/Sokoban.

6 PŘÍLOHA 1: UNITY PROJEKT

– **Unity Projekt.zip** – nezkompilovaný projekt pro Unity 2020.3.12f1

7 PŘÍLOHA 2: HRA

– **Hra.zip** – zkompilovaná a spustitelná hra