Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
Katedra informatiky

Bakalářská práce

# Tvorba HTML5 her s využitím frameworku MelonJS

# Programming HTML5 games using the MelonJS framework

**Vypracoval**: David Salcer
**Vedoucí práce**: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2015

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
**Fakulta pedagogická**
Akademický rok: **2013/2014**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **David SALCER** |
| Osobní číslo: | **P120264** |
| Studijní program: | **B7507 Specializace v pedagogice** |
| Studijní obor: | **Informační technologie a e-learning** |
| Název tématu: | **Programování HTML5 her s využitím frameworku MelonJS** |
| Zadávající katedra: | **Katedra informatiky** |

Zásady pro vypracování:

Cílem bakalářské práce je zpracování problematiky programování her v MelonJS frameworku, porovnání s konkurenčním frameworkem pro tvorbu her Phaser a prozkoumání dalších možností tvorby her na podobných principech. Student zpracuje podrobný český tutoriál k vytvoření vlastní hry pro Windows a MacOS resp. iOS, který bude sloužit IT studentům či vývojářům, zajímajícím se o aktuální počítačové technologie a programování her. Součástí práce bude provedeno i dotazníkové šetření za účelem zjištění, zda o této problematice mají studenti středních a vysokých škol informace či zájem. Praktickou částí bakalářské práce bude webová aplikace s vlastní desktopovou a mobilní hrou, vytvořenou v HTML5 a MelonJS včetně podrobné dokumentace a tutoriálu.

| | |
|---|---|
| Rozsah grafických prací: | CD ROM |
| Rozsah pracovní zprávy: | 40 |
| Forma zpracování bakalářské práce: | tištěná |

Seznam odborné literatury:


1. MelonJS: A lightweight HTML5 game engine. [online]. [cit. 2014-03-16]. Dostupné z: http://www.melonjs.org
2. Html5.cz: vše co potřebujete vědět o HTML5. [online]. [cit. 2014-03-16]. Dostupné z: http://www.html5.cz
3. Tiled: Map editor. [online]. [cit. 2014-03-16]. Dostupné z: http://www.mapeditor.org!
4. W3schools. [online]. [cit. 2014-03-16]. Dostupné z: http://www.w3schools.com/html/html5_intro.asp
5. GitHub: Build software better, together. [online]. [cit. 2014-03-16]. Dostupné z: https://github.com/melonjs/melonjs
6. THAU. Velký průvodce JavaScriptem: tvorba interaktivních webových stránek v praxi. 1. vyd. Praha: Grada, 2009. ISBN 978-80-247-2211-5
7. PEHLIVANIAN a NGUYEN. Velký průvodce JavaScriptem: tvorba interaktivních webových stránek v praxi. 1. vyd. Praha: Grada, 2009, 516 s. ISBN 9788025141632


| | |
|---|---|
| Vedoucí bakalářské práce: | **PaedDr. Petr Pexa, Ph.D.**<br>Katedra informatiky |
| Datum zadání bakalářské práce: | **27. března 2014** |
| Termín odevzdání bakalářské práce: | **30. dubna 2015** |

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 27. března 2014

# Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval/a samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 19. dubna 2015.

David Salcer

# Abstrakt

Cílem bakalářské práce je zpracování problematiky programování her v MelonJS frameworku, porovnání s konkurenčním frameworkem pro tvorbu her Phaser a prozkoumání dalších možností tvorby her na podobných principech. Student zpracuje podrobný český tutoriál k vytvoření vlastní hry pro Windows a MacOS resp. iOS, který bude sloužit IT studentům či vývojářům, zajímajícím se o aktuální počítačové technologie a programování her. Součástí práce bude provedeno i dotazníkové šetření za účelem zjištění, zda o této problematice mají studenti středních a vysokých škol informace či zájem.

Praktickou částí bakalářské práce bude webová aplikace s vlastní desktopovou a mobilní hrou, vytvořenou v HTML5 a MelonJS včetně podrobné dokumentace a tutoriálu.



## Klíčová Slova

HTML5, hry, MelonJS, javascript, CocoonJS, webový prohlížeč

## Abstract

The goal of this bachelor thesis is to get an overall picture of the process of programming games using the MelonJS framework. In addition, i will take the following steps to get a better view of this issue. Compare the MelonJS framework with its competitor, Phaser, and explore other ways to create games based on similar principles. Conduct a survey of high school and university students to determine awareness and interest in programming games. Create a public tutorial in Czech, for both Windows and Mac OS, that teaches students how to program their own games. The goal of this tutorial is to encourage students to pursue a career in computer studies. The ultimate goal is to publish the web and mobile game in a user-friendly manner on a free or hosted website. The game, documentation, and a tutorial would all be easily accessible from any internet-connected computer.



## Keywords

HTML5, games, MelonJS, javascript, CocoonJS, web browser

## Poděkování

Rád bych poděkoval vedoucímu mé závěrečné práce panu PaedDr. Petru Pexovi, Ph.D. za odborné vedení práce, rady, ochotu a vstřícnost při vedení mé práce.

Dále bych rád poděkoval mé rodině, která mi umožnila toto studium.

# Obsah

# 1 Úvod

Hraní her ve webovém prohlížeči bylo donedávna čistě věcí technologie Flash. s nástupem podpory HTML5 se ale tato technologie postupně vytrácí. Na poli her ve webovém rozhraní je vytlačována Javascriptem ve spojení s řadou volně šiřitelných, ale i placených herních frameworků, které využívají služeb dalších frameworků pro editaci grafiky, tvorbu prostředí, port na různá zařízení, distribuci do balíčků a přípravu pro online obchody jako je Apple Appstore, přídavné javascriptové moduly pro zavedení achievement-sytémů[1] pro hráče a spousta dalších vymožeností.

Práce je zaměřena na framework MelonJS a najdete v ní popis frameworku, využití, tutoriál k tvorbě hry, mou vlastní hru, seznam dalších frameworků a srovnání s konkurenčním Phaser frameworkem.

Výsledek práce bude prezentován ve webové aplikaci spolu s tutoriálem jak si naprogramovat vlastní hru na adrese: http://www.bp.davidsalcer.cz

## 1.1 Cíle práce

Cílem bakalářské práce je seznámení se s frameworkem MelonJS, popsání instalace frameworku na lokální počítač či webový server. Popsání základního principu fungování frameworku, sepsání co je třeba mít za software pro editaci prostředí hry a její spuštění ve webovém rozhraní popřípadě chytrém zařízení – smartphone či tablet.

Doporučené programy, které usnadňují práci, při tvorbě hry. Online zdroje, které poskytují zdarma grafiku, zvuky či hudbu pro tento typ her.

Sepsání jednotlivých kroků při tvorbě hry a problémy, které mohou vzniknout v jednotlivých krocích.

Vypsání přehledu frameworků, které lze při tvorbě her použít. Srovnání MelonJS s frameworkem Phaser, popřípadě vytvořit tak možnost pro další studenty navázat na tuto problematiku a využít právě konkurenčního frameworku Phaser k tvorbě dalších her.

[1] Achievement sytémy (Clay.io) umožňují bez změny kódu hry zpřístupnit různé úspěchy či bonusy ve

Vytvoření dotazníku za účelem zjištění informovanosti o pole tvorby HTML5 her a zájmu o tuto problematiku.

A závěrečná publikace návodu a vlastní hry na vlastní hosting.

## 1.2 Východiska práce

Mým cílem je tedy popsat práci s MelonJS. Popsat jednotlivé kroky při tvorbě hry, ty následně zapsat do formy návodu/tutoriálu s vlastní hrou a poté publikovat online. Herní frameworky jsou dobrým nástrojem, který usnadní práci při tvorbě hry, postará se o běh aplikace a my tak můžeme použít jeho funkce, k řešení našich konkrétních problémů při tvorbě hry. Zároveň nám dovoluje naprogramovat kód jednou, a za pomocí dalších nástrojů/služeb zpracovat naši hru tak, aby byla čitelná, i pro chytrá zařízení, popřípadě ji přímo nahrát k dispozici do online obchodů pro smartphony a tablety.

Protože frameworky prochází neustálým vývojem, stávají se tak čím dál více komplexnějším nástrojem. Lze tedy říci, že mohou složit nejen jako nástroj vzdělávání a učení se Javascriptu v praxi, či hobby programování jednodušších her, ale i jako nástroj pro začínající herní programátorský tým a vytvořit daleko složitější herní počiny jako jsou úspěšné hry Angry Birds[2] a Plants vs. Zombies[3].

[2] Angry Birds – Hra od společnosti Rovio Mobile http://cs.wikipedia.org/wiki/Angry_Birds

[3] Plant vs. Zombies – Hra od společnosti PopCap http://en.wikipedia.org/wiki/Plants_vs._Zombies

## 1.3 Metody práce

V úvodu práce bych chtěl uvést, co bude vše potřeba k tvorbě hry za software a kde ho získáme. Stručně seznámit s daným frameworkem a přejít k vlastnímu programování hry spolu s publikací návodu na webu, upozornit na problémy či chyby, které mohou nastat během jednotlivých kroků.

Vytvoření dotazníku na téma tvorby her pomocí těchto frameworků a pokusit se informovat a hlavně motivovat studenty ke studiu informatiky, dát nejen studentům možnost zkusit si naprogramovat svou hru doma a to za pomocí právě online dostupného webu s hrou, příslušnými daty a návodem.

# 2 Co je to framework?

Framework je obecně řečeno sada nástrojů, knihoven a procedur, které mají pomoci vývojářům nějakého projektu, aby se nemuseli zabývat řešením rutinních úkonů stále dokola, ale aby se soustředili na unikátní konkrétní problematiku projektu. Tyto nástroje či procedury jsou dostupné pomocí modulů a funkcí, které lze používat opakovaně a kdykoli je třeba. V našem případě framework MelonJS nabízí funkce rozeznání druhu zařízení (PC nebo mobilní zařízení), spuštění přehrávání hudby, grafické efekty, matematické funkce, komunikace se softwarovou vrstvou a další. Takže tyto funkce nemusíme vynalézat znovu, ale použijeme při řešení a vytváření naší hry.

# 3 MelonJS

MelonJS relativně nový, rychle se vyvíjející volně dostupný a bezplatný 2D open source herní engine, který spadá pod licenci MIT [1] , takže si každý může engine přizpůsobit podle svých požadavků úpravou jeho kódu a zároveň svojí logikou spadá pod HTML5 herní komunitu. Jedná se o samostatnou knihovnu, která na straně klienta nepotřebuje žádné instalace či výkonný hardware. Je závislá pouze na prohlížeči a jeho podpoře HTML5, třebaže z důvodu spuštění her v elementu `<canvas>` a více kanálového HTML5 audia či Web Audia.

Je vyvíjený od roku 2011 třemi vývojáři, Oliver Biot, Jason Oster a Aron McLeod, kteří jsou běžně k zastižení na Gitter chatu [2] .

Framework je kompatibilní se všemi majoritními prohlížeči (Chrome, Safari, Firefox, Opera a Internet Explorer), zároveň je schopný běhu na mobilních zařízeních i s využitím jejich gyroskopu, vysokého DPI[4], automatického překlápění zařízení a podpory dotykového displeje.

V základu je implementována základní fyzika, podpora systémových a bitmapových fontů, pohyby a přechody mezi objekty, změn obrazovek, výpočet kolize objektů a rozšířená podpora matematických funkcí.

## 3.1 Podpora

Protože se jedná o volně a zdarma dostupný framework, tak tomu náležitě odpovídá i podpora, která záleží na dobrovolnících a komunitách. Samotný hlavní vývojový tým čítá zhruba tucet nadšenců, kteří pomocí online verzovacího systému sdílí a vyvíjí MelonJS. Existuje ale prostor pro ohlašování chyb, návrhů zlepšení a i jakási poradna. Rychlost vyřešení či reakce na vámi poslaný požadavek je různá, ale není zaručená.

Zároveň je nutno podotknout, že podpora je čistě v anglickém jazyce. Žádné české komunity, fóra, tutoriály atp. zatím neexistují.

[4] Dots per inch – obrazových bodů či pixelů na délku jednoho palce

Podporu lze hledat na těchto webových stránkách:

- Issue tracker: https://github.com/melonjs/melonJS/issues
- MelonJS forum: http://melonjs.org/forum.html
- Gitter chat: https://gitter.im/melonjs/public
- MelonJS Wiki: https://github.com/melonjs/melonJS/wiki

## 3.2 Instalace frameworku MelonJS

Protože se jedná o javascriptové knihovny, žádnou klasickou instalaci nepotřebují. Na oficiálních stránkách frameworku je vždy ke stažení poslední schválená a odzkoušená verze jádra. Stáhnout lze jak verzi pro vývoj nezmenšenou, tak verzi pro ty, kteří nebudou editovat zdrojový kód enginu.

V našem případě opravdu bude stačit verze zmenšená, protože se nebudeme zabývat laděním a vývojem jádra, budeme ho pouze využívat. Rozdíl ve velikosti je pak znatelný. Nezmenšená verze pro vývoj má v aktuální verzi 2.0.2, 772 kilobajtů a verze zmenšená pak pouhých 168 kilobajtů.

Verze pro vývoj, lze samozřejmě stáhnout a editovat v jakémkoli textovém procesoru, ale hlavně je určena pro instalaci přes příkazový řádek a následné sestavení vlastní verze. v tomto případě se nás to ale netýká.

Pro lepší a přehlednější začátek tvorby jsem zvolil jinou cestu, nežli stažení samotné knihovny. Podobně jako u tvorby webových prezentací existuje tzv. boilerplate, který nám usnadní začátek, zpřehlední manipulaci a práci s projektem tvorby hry.

Odkaz lze nalézt samozřejmě na oficiálních stránkách a na online repositáři GitHub https://github.com/melonjs/boilerplate .

V komprimovaném zip balíčku, který si stáhneme, se nachází optimálně uspořádané složky, aby nám to usnadnilo práci a rozložení projektu. Obecně zde najdeme obě verze enginu, přípravu pro obrazovky jako je `title.js` a `play.js`, soubory pro zprávu herních objektů `entities.js`, soubor pro načtení HUD[5] prvků `HUD.js`,

[5] Head-up displej – zobrazování informací o průběhu hry, stavu hráče, mapy apod.

soubor pro inicializaci hry `game.js`, soubor do kterého budeme uvádět co vše se má do hry načíst za soubory `resources.js`.

A hlavně zde najdeme spouštěcí soubor `index.html`, což je vlastně klasický HTML soubor s webovou prezentací, která má v hlavičce importované jednotlivé soubory výše zmíněné.

# 4 Další herní frameworky

Nejdříve se zaměřím na tvorbu hry hlavně v MelonJS, v závěru práce jej pak srovnám s dalším velice oblíbeným frameworkem Phaser. Je nutné ale podotknout, pro ty, kteří by se zajímali o tvorbu her, že v současnosti existují další herní frameworky na podobné bázi – javascript, HTML5 a CSS.

Je tedy rozhodně z čeho vybírat. Lze vybírat třeba i na základě informací z webu Clay.io, který vypracoval seznam frameworků a jejich oblíbeností, na základě monitoringu svých služeb a modulů pro tyto frameworky.

| **Název** | **Cena** | **Popularita (0 až 100)** | **Hodnocení (0 až 5)** |
|---|---|---|---|
| Construct 2 | Různé varianty | 100 | 4 |
| ImpactJS | $99 | 86 | 4 |
| EaselJS | Zdarma (MIT) | 63 | 4,5 |
| Pixi.JS | Zdarma (MIT) | 55 | 5 |
| Phaser | Zdarma (MIT) | 54 | 4,5 |
| GameMaker | Různé varianty | 52 | 4 |
| Three.js | Zdarma (MIT) | 51 | 4,5 |
| PlayCanvas | Zdarma | 49 | 4 |
| Turbulenz | Zdarma (MIT) | 48 | 4 |
| lycheeJS | Zdarma (MIT) | 43 | 4,5 |
| MelonJS | Zdarma (MIT) | 41 | 4,5 |

| | | | |
|---|---|---|---|
| Cocos2d-X | Zdarma (MIT) | 40 | 4 |
| Quintus | Zdarma (MIT) | 40 | 4 |
| WADE | Různé varianty | 39 | 4,5 |
| Crafty | Zdarma (MIT) | 37 | 4,5 |
| LimeJS | Zdarma (Apache) | 35 | 4 |
| Enchant.js | Zdarma (MIT) | 33 | 3 |
| Isogenic Engine | Různé varianty | 30 | 4 |
| GC DevKit | Zdarma (Mozilla) | 29 | 4 |
| Voxel.js | Zdarma (BSD) | 26 | 4 |
| Panda.js | Zdarma (MIT) | 21 | 4 |
| Kiwi.js | Zdarma (MIT) | 20 | 3,5 |

**Tabulka č. 1 –** Seznam populárních frameworků dle webu Clay.io [4]

## 4.1 Podpora herních frameworků

Zde je mnou vytvořený hrubý přehled podpory herních frameworků. Prošel jsem webové stránky těchto frameworků a procházel, co vše nabízí. Protože většina těchto frameworků je zcela zdarma, tak podporu své komunity dělí mezi několik dalších služeb jako je google+, github a bezplatná fóra. Některé dokonce nemají ani své webové stránky a poskytují framework jako doplněk k jiným a rozšířenějším frameworkům viz.: Pixi.js, který je čistě grafický engine a je kupříkladu zcela zabudován do Phaseru.

Pokud bychom chtěli zjistit, jak početná je komunita jednotlivých frameworků, tak neuspějeme. Pouze jediný Construct2 má na svých stránkách uveden počet stažení frameworku, který se blíží ke dvěma miliónům a na fóru je zaregistrováno přes 125 tisíc uživatelů.

Dále počty tutoriálů a příkladů jsou různé. Zde asi vede framework Phaser, který má opravdu velké množství příkladů oproti ostatním.

Jedna z dobrých zpráv, co mají všechny frameworky společné je datum poslední aktualizace, které vesměs není starší jak 5 měsíců od aktuálního data.

| **Název** | **Programovací jazyk** | **Podpora** | **Poslední aktualizace** |
|---|---|---|---|
| Construct2 | Žádný | Web, fórum, blog, tutoriály, manuál, dokumentace, knihy, obchod s doplňky, bety | Srpen 2014 |
| ImpactJS | Javascript | Web, fórum, blog, knihy | Červenec 2014 |
| EaselJS | Javascript | Web, fórum, blog, GitHub, google+, stackoverflow | Prosinec 2014 |
| Pixi.js | Javascript | GitHub, tutoriály, dokumentace, | Leden 2015 |
| Phaser | Javascript, Typescript | Github, tutoriály, dokumentace, fórum, knihy, příklady | Leden 2015 |
| GameMaker | Javascript | Tutoriály, vzdělávací program, projekty, dokumentace, helpdesk | Srpen 2014 |
| Three.js | Javascript | GitHub, wiki, online editor, google+, příklady, chat, dokumentace, stackoverflow, knihy | Leden 2015 |
| PlayCanvas | Javascript | Online editor, hosting zdarma, GitHub, fórum, příklady, | Únor 2015 |
| Turbulenz | Javascript | Sociální sítě, dokumentace, příklady, vlastní repozitář | Září 2014 |

| | | | |
|---|---|---|---|
| lycheeJS | Javascript | GitHub, tutoriály, příklady, facebook, youtube | Leden 2015 |
| melonJS | Javascript | GitHub, google+, chat, bety, dokumentace | Únor 2015 |

**Tabulka č. 2 –** Tabulka podpory herních frameworků

# 5 Tvorba hry

## 5.1 Programy a zdroje

Z hlediska instalace frameworku MelonJS je tedy vše hotové pouhým zkopírováním souborů. Opravdu instalovat budeme a to zejména kvůli práci s tímto frameworkem. Budeme potřebovat editovat javascript pokud možno efektivně s nápovědou kódu a doplňováním syntaxe, dále bude potřeba stáhnout program Tiled, který se bude starat o tvorbu herních prostředí, map a jejich editaci.

Pro editování javascript potažmo HTML kódu jsem si vybral volně stažitelný a multiplatformní program Brackets, který stáhneme na této oficiální adrese: http://brackets.io/.

Program Tiled, též volně stažitelný a multiplatformní je ke stažení na této oficiální adrese: http://www.mapeditor.org/.

Dále je potřeba myslet na grafiku, hudbu a zvukové efekty. Pokud nepracujeme v týmu, budeme sami zastávat funkci několika osob najednou. Bohužel tvorba hlavního hrdiny, či nepřátel bývá náročná na grafický um, jsou zde možnosti jak se tomuto kroku vyhnout. Doporučuji tedy stránky http://opengameart.org, které obsahují nepřeberné množství grafiky určené pro hry. Vyhledávání v databázi obsahuje filtr licencí, budeme tedy hledat takové výtvory, které jsou kupříkladu pod licencí Public domain.

To samé lze aplikovat i na hledání hudby a zvukových efektů do naší hry, kupříkladu https://www.freesound.org/.

## 5.2 Než začneme

Než začneme s tvorbou hry, budeme potřebovat následující věci:

- Tiled map editor, který je k dispozici zde: www.mapeditor.org
- MelonJS ve verzi 2.0.2, kterou stáhnete z mých stránek www.bp.davidsalcer.cz a nebo na www.melonjs.com
- A data pro tutoriál. Obrázky hráče, nepřátel, lektvarů apod., opět ke stažení na www.bp.davidsalcer.cz
- Při tvorbě se můžete dívat do dokumentace, kterou najdete na www.melonjs.com a zkrácenou verzi na www.bp.davidsalcer.cz

## 5.3 Před spuštěním hry

Pro spuštění hry budeme samozřejmě potřebovat prohlížeč, který podporuje HTML5. v dnešní době tedy každý majoritní prohlížeč (Safari, Chrome, Firefox i Internet Explorer). Osobně jsem si při tvorbě hry vybral Safari, ale vřele doporučuji prohlížeč Chrome od Googlu, který je velice rychlý s MelonJS frameworkem.

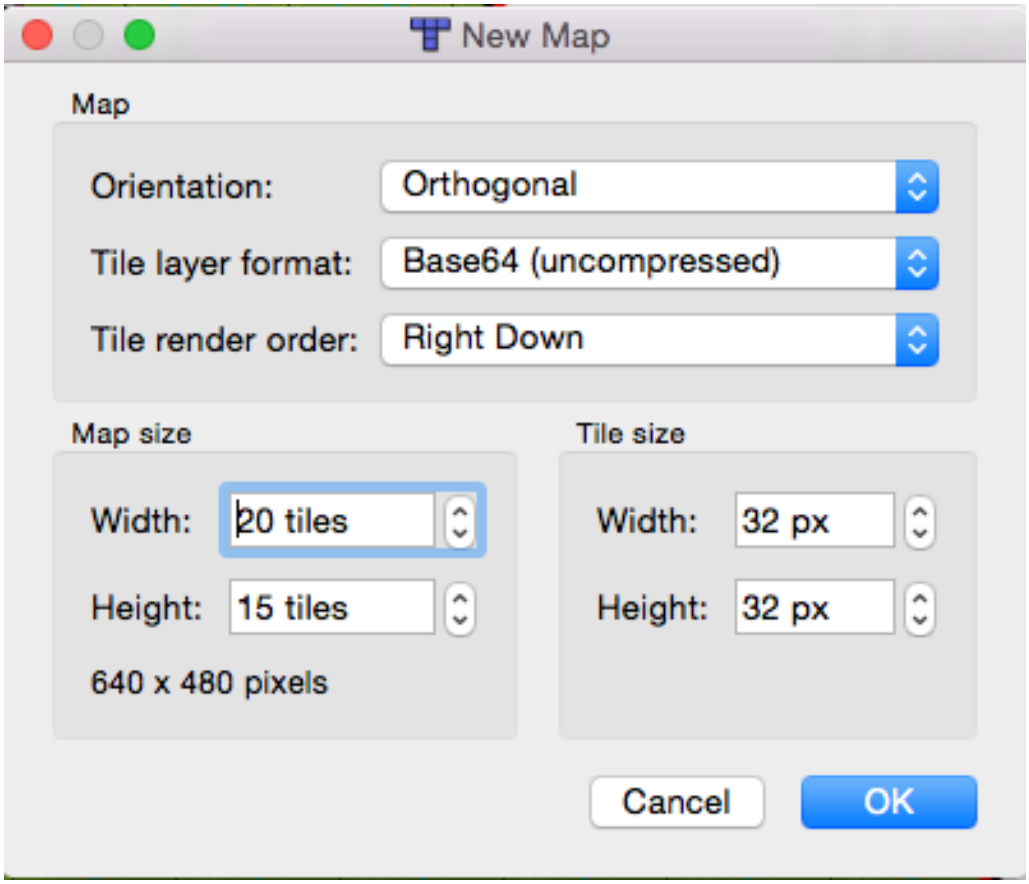
Protože, budeme testovat hru lokálně na našem počítači, před spuštěním hry, musíme zapnout vývojářské nástroje (developer tools) a povolit, či spíše omezit ochranu proti lokálním restrikcím či omezením.

Každý prohlížeč to má trochu pod jiným názvem, v Safari se jedná o „Disable local file restrictions“, v Chromu to takto bohužel nejde a musíte program spouštět s parametrem `--disable-web-security` či lepší `--allow-file-access-from-files.` u prohlížeče Firefox toto řešit nemusíme, ten funguje bez jakýchkoli omezení.

Další možností je, že hru budeme testovat na lokálním web serveru, nebo tedy přímo na webserveru nějakého providera př.: Active 24, Wedos apod.

# 6 Vytváříme level v Tiled

Nyní si tedy můžeme otevřít Tiled map editor a začít s tvorbou prvního levelu, lokace, mapy apod. Budeme tvořit 2D hru, tzv. plošinovku či skákačku, podle vzoru legendární hry Super Mario, kterou jistě mnozí znáte.




**Obrázek 1 –** Vytváříme mapu v Tiled

Zde si vybíráme orientaci levelu. Na výběr máme Orthogonal[6], Isometric a Hexagonal. Pro účel 2D hry z pohledu ze strany či ze shora je zde právě pravidelné rozložení sítě (grid) – Orthogonal. Jako podklad používá MelonJS nekompresovaný Base64, jak vidíte na obrázku. S jinou metodou by hra nenačítala level. Dále metoda jak se bude mapa renderovat (vykreslovat), v jakém pořadí, můžeme nechat v základním nastavení shora dolů.

Nyní přicházíme k zajmavějšímu nastavení. Velikost mapy (map size) a velikost dlaždic (tile size). Na velikosti mapy až tolik nezáleží, Melon si je schopný poradit

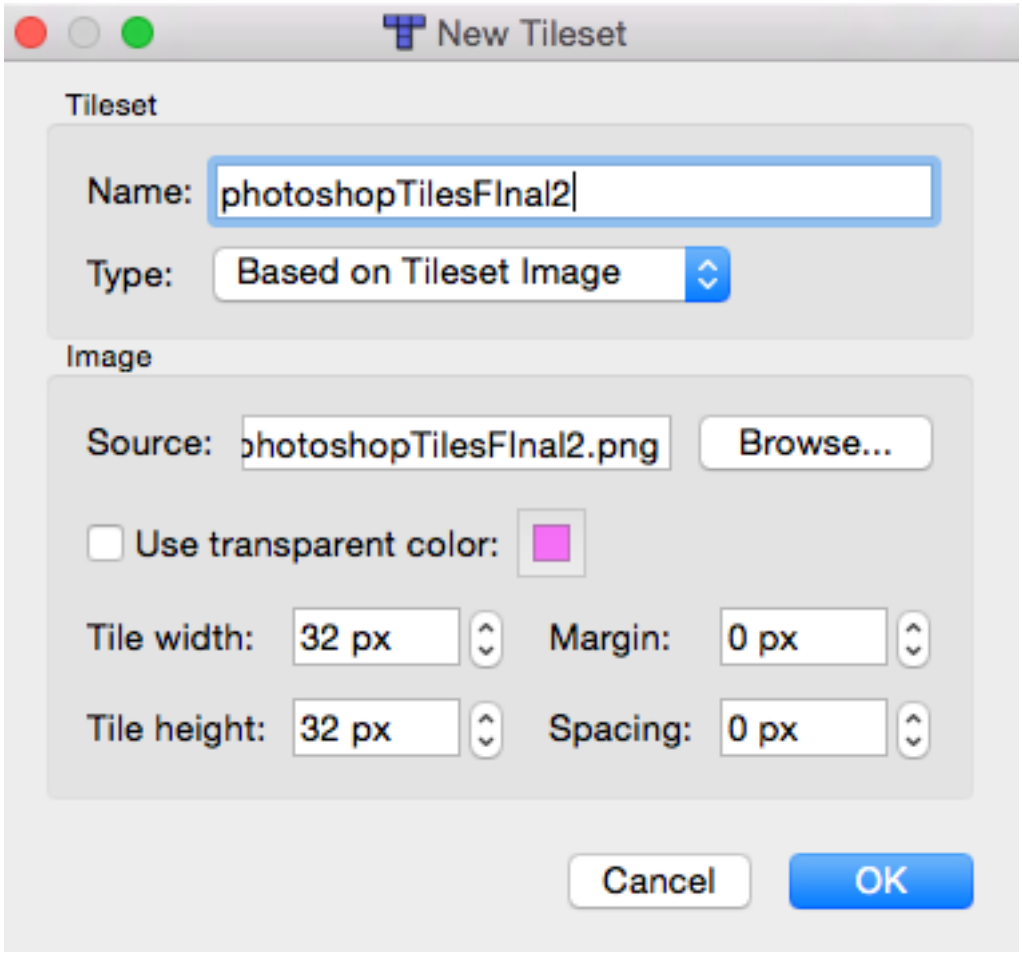
[6] Kolmý, ortogonální

i s velkou mapou a centruje kameru na hráče, mapu následně posouvá dál. Důležitější je velikost dlaždic. To znamená, že se nám herní pole rozřeže na čtvercové dlaždice o velikosti 32x32 pixelů. Až později nahrajeme obrázek pro podklad mapy „Tileset“, tak se nám rozřeže podle tohoto nastavení mapy. Pokud bychom tedy, s tímto nastavením nepočítali, obrázek by se nám rozřezal špatně. Fungovalo by to, ale level by nebyl hezký a z tilesetu bychom nemohli udělat level tak, jak se patří.

Osobně si myslím, že velikost 32x32 pixelů je optimální. Logicky lze docílit i naoko lepší grafiky tím, že zmenšíme velikost dlaždic kupříkladu na 16x16 pixelů. Sázení dlaždic do mapy je tak zdlouhavější, ale grafika se při stejném rozlišení zdá být hezčí a více detailní.

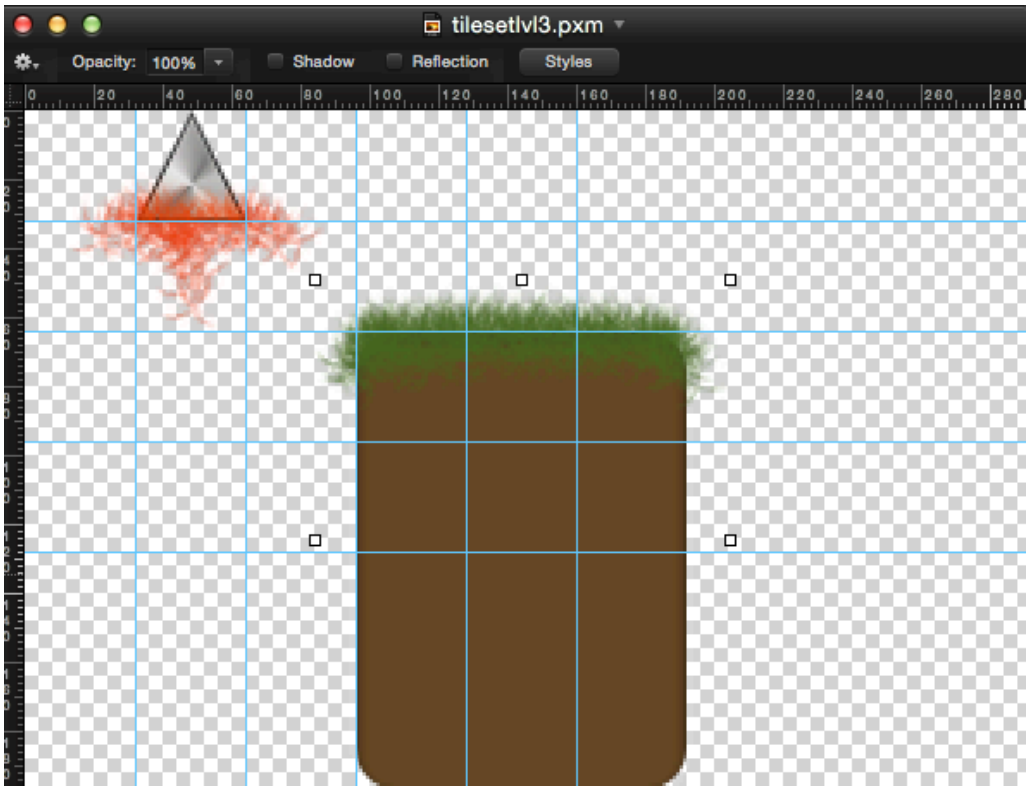
## 6.1 Nahráváme tileset

Jak jsem již zmínil, termín „tileset“ tedy znamená podklad pro mapu, či ve volném překladu set dlaždic. v Tiledu vidíme prázdný prostor, pouze s viditelnou mřížkou. Nový tileset nahrajeme přes navigaci „Map – New tileset“.




**Obrázek 2 –** Nahrávání tilesetu do mapy

Pomocí „Browse“ si nahrajeme tileset, který potřebujeme. Zde bych se nastavením nezaobíral. Pouze zmíním, že lze využít funkce „use transparent color“, což se běžně používá při tvorbě takových her. Celý podklad je obklopen dejme tomu růžovou barvou a je formátu JPG, lze si jej pomocí této funkce odfiltrovat a udělat z něj prakticky formát PNG. Dále jsou zde k doladění možnosti okrajů (margin) a mezer (spacing). Pokud by se stalo, že máme tileset trochu nesouměrný, či chybí pár pixelů, lze tyto nedostatky doladit. Tileset s tímto názvem si můžete stáhnout z mých stránek, lze si ale udělat vlastní, třeba v programu Pixelmator (pro Mac OS). Použijte funkce pravítek, u grafických editorů, abyste docílili správného rozřezání tilesetu (32x32px).



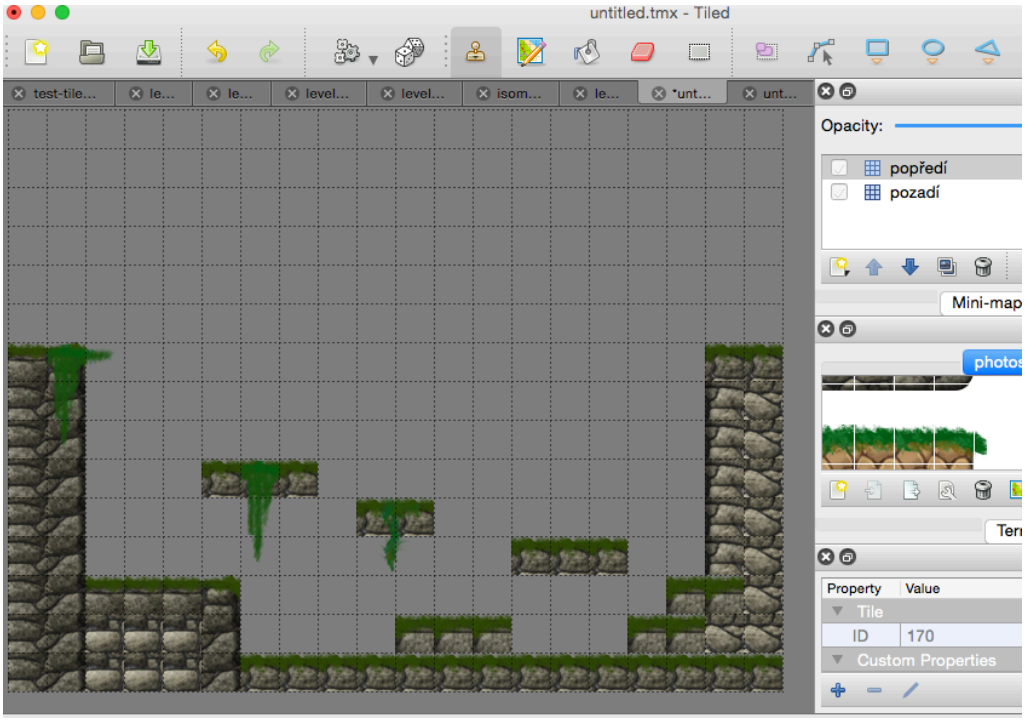

**Obrázek 3 –** Příklad rozvržení tilesetu v Pixelmatoru

## 6.2 Vytváříme prostředí (level)

Nyní si vytvoříme dvě dlaždicové vrstvy (tile layer) „popředí“ a „pozadí“. Ve vrstvě „pozadí“ budeme vytvářet to, po čem bude hráč chodit, tedy pevný podklad a na něj budeme aplikovat kolizi později. Kdežto na vrstvu „popředí“ nebudeme aplikovat nic, ta bude sloužit pouze k doladění detailů levelu. Mohou to být značky, květiny, efekty, překryvy, prakticky cokoli. Tato vrstva nebude reagovat na hráče, podle umístění se budou s hráčem překrývat.

Touto vrstvou „popředí“ můžeme docílit efektu, že hráč chodí v trávě. Pokud tomu přizpůsobíme tileset. (pro tento efekt je dobrý tileset s názvem tilesetlvl3.png)

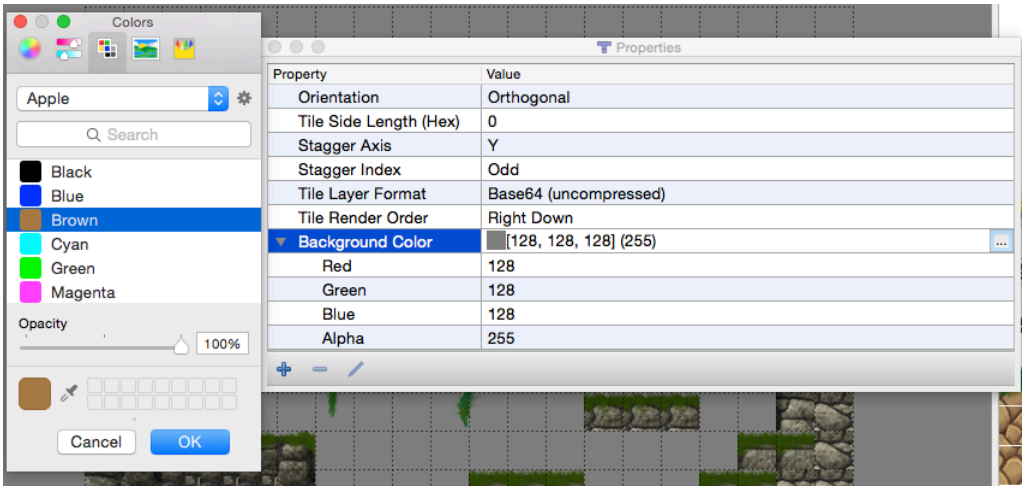
Tiled si pamatuje absolutní cesty k tileset souborům, je tedy dobré vytvářet projekt na jednom místě a nepřenášet soubory, jinak se může stát, že level znovu k editaci neotevřeme.




**Obrázek 4 –** Tvorba levelu a vrstev

Na obrázku lze tedy vidět, že pevnou kostru levelu kreslíme do vrstvy „pozadí“ a ostatní věci, které nebudou pevné, do vrstvy „popředí“ (na obrázku liány).

Nastavíme si pozadí levelu přes „Map – Map properities“, třeba na hnědou barvu.




**Obrázek 5 –** Nastavení barevného pozadí levelu

Nyní naší mapu uložíme jako (Save As) soubor s příponou TMX a přejdeme k dalšímu bodu.

## 6.3 První načtení prázdného levelu

Nyní si otevřeme soubor game.js. Funkce `onload()` se poastará, že až se HTML stránka načte, začne provádět následující kód. Funkce `me.video.init()`, se pak postará, že do HTML elementu s id „screen“ vytvoří HMTL5 element `<canvas>` o velikosti 853x480 pixelů. Dalším atributem s hodnotou „true“ nastavíme dvojité bufferování (více než jedna vyrovnávací pamět) a hodnota „auto“ nám zajistí automatickou velikost okna, aby se velikost přizpůsobila velikosti displeje zařízení. Kód by měl vypadat takto:

```
var game = {

data : { score: 0, life: 1},

onload : function () {

        if   (!me.video.init('screen',   me.video.CANVAS,
853, 480, true, 'auto')) {

            alert("Your browser does not support HTML5
canvas.");

            return; }

        if (document.location.hash === "#debug") {

            window.onReady(function () {


me.plugin.register.defer(this,me.debug.Panel,"debug",
me.input.KEY.V);

            });}

        me.audio.init("mp3,ogg");

        me.loader.preload(game.resources);

        me.loader.onload = this.loaded.bind(this);

        me.state.change(me.state.LOADING);

    },
```

```
    loaded : function () {
    me.state.set(me.state.PLAY,newgame.PlayScreen());
        me.state.change(me.state.PLAY);
    }
};
```

Dále přidáme tileset a vytvořený level do objektu `resources`, ze kterého hra načítá soubory v něm uvedené. Otevřeme si tedy soubor resources.js a přidáme následující řádky:

```
game.resources = [
     {name:   "photoshopTilesFInal2",      type:"image",
     src: "data/img/map/photoshopTilesFInal2.png"},
     {name: "level1",              type: "tmx",  src:
"data/map/level1.tmx"},
];
```

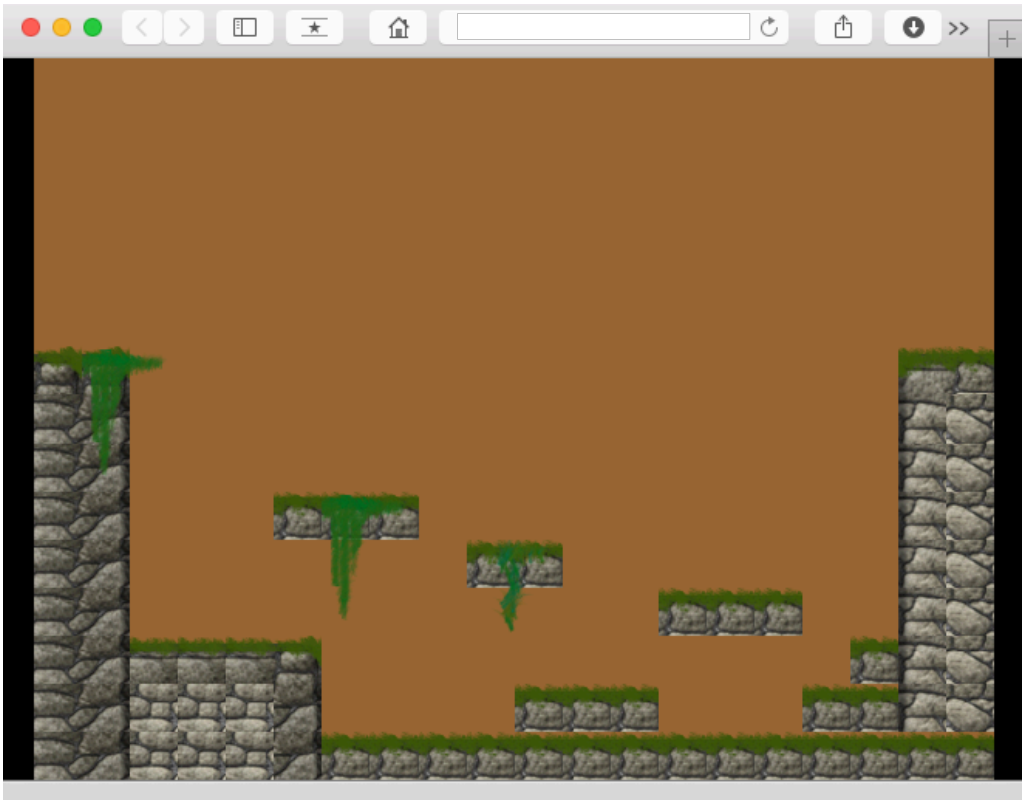
Jedná se vlastně o pole souborů, které se budou načítat. Zde uvádíme jméno souboru (name), typ souboru (type) a lokaci souboru (src).

Poslední částí je spuštění levelu v souboru play.js:

```
game.PlayScreen = me.ScreenObject.extend({
     onResetEvent: function() {
          me.levelDirector.loadLevel("level1");
     },
     onDestroyEvent: function() {
     }
});
```

Zkontrolujeme, zda ve spouštěcím souboru index.html máme připojené všechny soubory.

```
<script src="lib/melonJS-2.0.2-min.js"> </script>
<script src="js/game.js"></script>
<script src="js/resources.js"></script>
<script src="js/entities/entities.js"></script>
<script src="js/screens/play.js"></script>
```



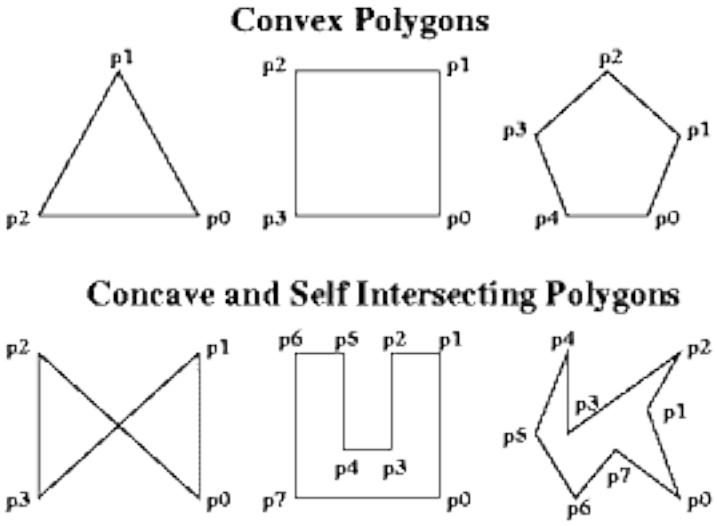
**Obrázek 6** – První spuštění

A takto by to mělo vypadat po prvním načtení prázdného levelu.

## 6.4 Kolizní vrstva

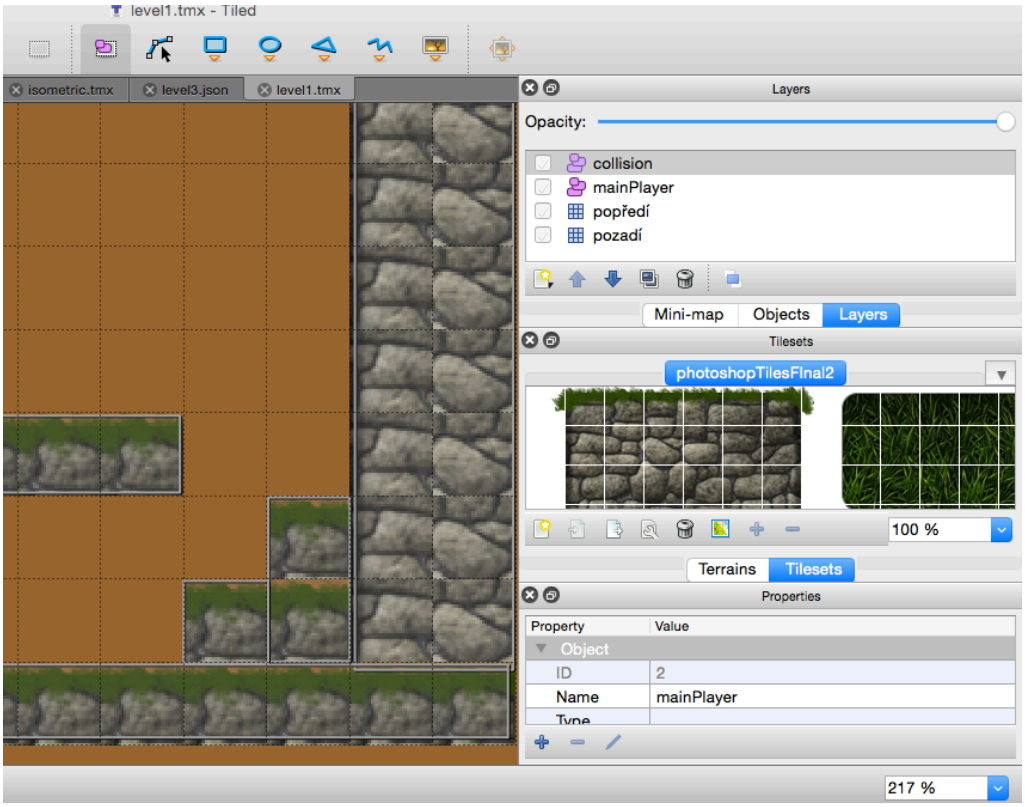
Kdybychom teď přidali hlavního hráče, postava by se sice zobrazila, ale během pár sekund by propadnul pryč z obrazovky. Nyní se tedy musí založit nová vrstva v Tiled editoru, která bude sloužit pro kolize. MelonJS tak rozezná pevný povrch od nepevného. Dřívější verze MelonJS toto řešily přes klasický tile layer, ve kterém byly tzv. meta tileset či dlaždice, které se skládaly pod dlaždice, které měly být pevné. Novější verze toto již řeší přes objektovou vrstvu, která musí obsahovat slovo „collision“. K rozeznání využívá algoritmus separace os (Separating axis theorem

algorythm [5]). To znamená, že objekty ve vrstvě „collision“ mohou být pouze konvexní polygony nikoli konkávní. Vnitřní úhly nesmí být více jak 180 stupňů a hrany se nesmí křížit. Mimo těchto objektů, lze vytvářet i úsečky jako kolizní podklad, třeba k platformám, zdím apod.




**Obrázek 7 –** Konvexní a konkávní polygony [6]

Vytvoříme si tedy objektovou vrstvu nazvanou „collision“ a pomocí nástrojů v horní liště programu obkreslíme náš terén, který bude pevný. (Insert rectangle tool). Při kreslení objektů je nutné dávat pozor, jakou objektovou vrstvu máme vybranou v pravém okně „vrstvy“ (layers). Ta, která je zvýrazněná, do té kreslíme objekty.




**Obrázek 8 –** Kreslení kolizí

Více zatím není třeba definovat. Můžeme pokračovat přidáním hráče.

## 6.5 JSON vs. TMX

Program Tiled zároveň umožňuje ukládání levelu do formátu JSON, který se načítá při spouštění hry rychleji, než ve formátu TMX. Pokud si otevřeme vytvořený level například v programu Brackets, zjistíme, že TMX formát má strukturu XML souboru. Obsahuje spoustu elementů a mnoho z nich má i atributy apod. Pokud uložíme tentýž level do formátu JSON, zjistíme, že se jedná o jeden javascriptový objekt.

Level ve formátu JSON nemusí být nutně menší, co se týče velikosti KB, ale parsování tohoto souboru je pro MelonJS rychlejší, takže i takto můžeme docílit případného zrychlení načítání hry.

Pokud si zvolíme ukládání levelů do formátu JSON, pak samozřejmě změníme koncovky dosavadních levelů z TMX na JSON v resources.js.

```
{name: "level1",                type: "tmx",  src:
"data/map/level1.json"},
```

Zde musíme dávat pozor na atribut „type“, který zůstává stejný. Tento atribut znamená pro framework pouze to, že se bude jednat o načítání mapy/levelu, nikoli typ souboru. Tuto informaci mi poskytli vývojáři MelonuJS, kteří sami uznali, že je tento atribut matoucí a v dalších verzích frameworku se na toto také zaměří.

# 7 Přidání hlavního hráče

Podobně jako se rozřezává tileset, tak zde budeme na podobném principu pracovat s obrázkem hlavního hráče s tím rozdílem, že se může jednat o jeden dlouhý soubor PNG (jednořádkový) a nebo několik souborů zvlášť pro každou animaci, kterou budeme později definovat.

V našem případě máme k dispozici jednořádkový grafický soubor s hlavním hráčem student_hero_fall.png.

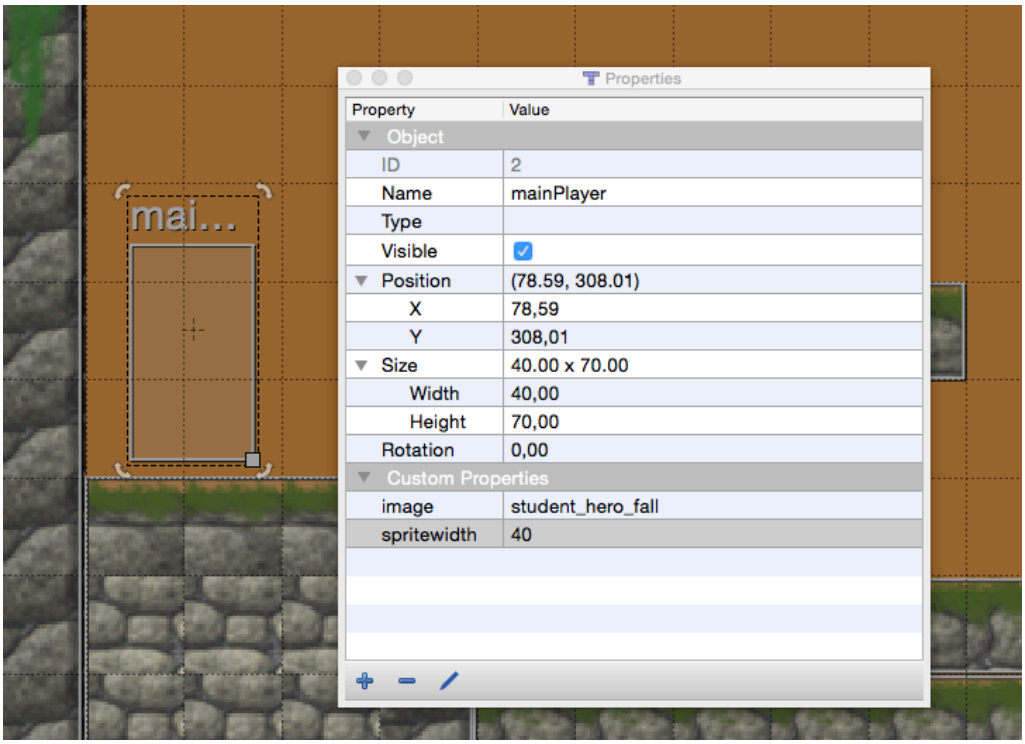
**Obrázek 9 –** Hlavní hráč

Takovéto soubory můžeme stáhnout z internetu a říká se jim „sprites“ či „spritesheet“.

Obrázek vytvářet podobným způsobem jako se vytváří tilesety. Zde je důležitá šířka jednoho obrázku hráče. Zde 40 pixelů šířka a 70 pixelů výška. Hra bude animovat hráčův pohyb tím, že bude okénkem 40x70 pixelů zobrazovat jednotlivé pozice ze souboru student_hero_fall.png. Zde je nutné si dávat pozor, aby vše odpovídalo. Výška hráče v Tiled musí teda odpovídat výšce souboru. Pokud tedy uvedeme šířku našeho hráče 40 pixelů, musí tedy celkový soubor PNG být dělitelný 40ti a to beze zbytku.

Začneme tím, že v Tiledu si vytvoříme novou objektovou vrstvu „mainPlayer“ a do té vrstvy vytvoříme nový objekt, kterému nastavíme následující hodnoty a přesuneme jej na místo, kde chceme, aby se naše postava hráče zjevila.

Tento objekt pojmenujeme stejně jako vrstvu – „mainPlayer“ a nastavíme velikost (size). Přičemž šířka (width) bude 40 a výška (height) 70 (pixelů). Dále musíme přidat speciální atributy a to atribut „image“, kterému nastavíme hodnotu takovou jako je jméno obrázku, z kterého načítáme obrázek hráče (student_hero_fall). A poslední hodnotu „spritewidth“, což je šířka našeho hráče, na 40 (pixelů).




**Obrázek 10** – Nastavení objektu mainPlayer

## 7.1 Kód hlavní postavy hráče

Začneme tím, že přidáme obrázek hlavního hráče do pole resources.js.

```
{name: "student_hero_fall", type:"image", src:
"data/img/sprite/student_hero_fall.png" },
```

Dále přidáme kód do entities.js. Zde založíme konstruktor hráče a od teď se v něm bude přidávat a upravovat veškerý kód týkající se hráče. MelonJS vytváří několik typů herních objektů, objekty hráče a nepřátel jsou kupříkladu typu me.Entity [22], který má nějaké základní vlastnosti a ty my rozšiřujeme, proto me.Entity.extend (podobně jako v jazyce Java, třídy extend).

Prázdný kód vypadá takto:

```
game.PlayerEntity = me.Entity.extend({

});
```

A dovnitř tohoto objektu budeme přidávat funkce. Logika MelonJS je taková že, každý objekt typu me.Entity začíná svojí „init“ funkcí tj. Inicializace objektu:

```
init:function (x, y, settings) {
```

```
    this._super(me.Entity, 'init', [x, y , settings]);

    this.body.setVelocity(6, 15);

    me.game.viewport.follow(this.pos,
me.game.viewport.AXIS.BOTH);

    this.alwaysUpdate = true;

        // animace

        this.renderable.addAnimation("walk",  [3, 4, 5,
6, 7, 8, 9]);

        this.renderable.addAnimation("stand",
[0,0,0,0,0,0,0,0,1,1,1,1,1,1]);

        this.renderable.addAnimation("jump", [2]);

        this.renderable.addAnimation("fall", [10]);

        this.renderable.setCurrentAnimation("stand");

        this.body.collisionType =
me.collision.types.PLAYER_OBJECT;

    },
```

Zde zavoláme konstruktor this._super a odkážeme na první inicializační funkci „init“. Dále nastavíme hráčovu rychlost chození a skákání (setVelocity), nastavíme obrazovku tak, aby následovala hráče podle obou os (viewport.follow, AXIS.BOTH).

A dojde i na zmíněné animace. Zde přidáme několik animací podle našeho obrázku hráče. Pokud si rozdělíme obrázek na 11 částí po 40 pixelech a očíslujeme od 0 do 10, pak snadno pochopíme o co se jedná. Do funkce `this.renderable.addAnimation()` napíšeme jaké jméno má animace mít a pořadí či pozice částí našeho obrázku. Zde kupříkladu animace chození (walk), se bude skládat ze za sebou jdoucích pozic 3 až 9.

Dále je na řade funkce update, která se volá 60 krát za sekundu (60 FPS) a kontroluje stav hráče, zda není v kolizi s jiným objektem či jestli jsme stiskli tlačítko pro pohyb hráče.

```
update : function (dt) {
```

```
if (me.input.isKeyPressed('left')){
    this.renderable.flipX(true);
    this.body.vel.x -= this.body.accel.x * me.timer.tick;
     if (!this.renderable.isCurrentAnimation("walk")) {
          this.renderable.setCurrentAnimation("walk");
     }
}
else if (me.input.isKeyPressed('right')){
     this.renderable.flipX(false);
            this.body.vel.x += this.body.accel.x *
me.timer.tick;
if (!this.renderable.isCurrentAnimation("walk")) {

this.renderable.setCurrentAnimation("walk");
}
}

 else{    this.body.vel.x = 0;
     if( !this.renderable.isCurrentAnimation("stand")) {
this.renderable.setCurrentAnimation("stand");
            }
        }
        if (me.input.isKeyPressed('jump'))
        {
if (!this.body.jumping && !this.body.falling) {
this.body.vel.y = -this.body.maxVel.y * me.timer.tick;
this.body.jumping = true;
```

```
me.audio.play("jump");

}

}

if(this.body.jumping){

this.renderable.setCurrentAnimation("jump");

}

else if(this.body.falling){

this.renderable.setCurrentAnimation("fall");

}

this.body.update(dt);

me.collision.check(this);


return (this._super(me.Entity, 'update', [dt]) ||
this.body.vel.x !== 0 || this.body.vel.y !== 0);

},
```

Poslední základní funkce je funkce `onCollision()`, která se volá v případě, že se hráč dotýká jiného objektu. Zde právě později připíšeme, co vše se má stát, pokud je postava hráče v kolizi.

```
onCollision : function (response, other) {return true;}
```

Kód postavy hráče je hotový. Nyní zbývá už jen inicializovat (namapovat) tlačítka z klávesnice, aby MelonJS očekával jejich stisknutí. Dále naši postavu přidáme (zaregistrujeme) do herního světa (me.pool.registr).

Proto se vrátíme do game.js a doplníme do funkce `loaded()` následující rádky:

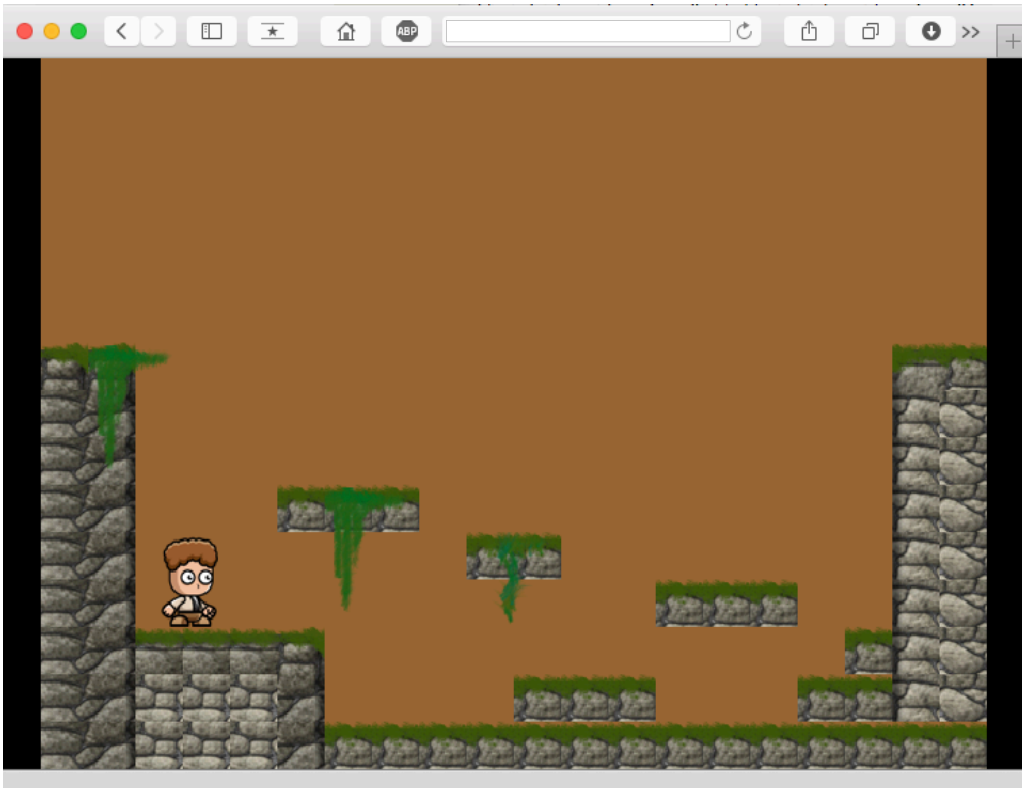
```
me.pool.register("mainPlayer", game.PlayerEntity);
```

Mapování tlačítek, doleva (LEFT), doprava (RIGHT) a skok (X) apod.

```
me.input.bindKey(me.input.KEY.LEFT,  "left");

me.input.bindKey(me.input.KEY.RIGHT, "right");
```

```
me.input.bindKey(me.input.KEY.UP, "jump");
me.input.bindKey(me.input.KEY.DOWN, "down");
me.input.bindKey(me.input.KEY.SPACE, "jump");
me.input.bindKey(me.input.KEY.X,     "jump", true);
```

Zde si můžeme namapovat i více tlačítek pro jednu funkci, tím že je pojmenujeme stejně. Třeba pro skákání (jump) je zde přiřazen mezerník, šipka nahoru, ale i klávesa X. u tlačítek pro skákání musíme uvést ještě druhý parametr s hodnotou „true“. Ten zajistí, aby se při stisku klávesy ihned deaktivoval, aby se tedy nestalo, že hráč odskáče z obrazovky nahoru. U směrových kláves (doprava, doleva) toto nepotřebujeme, tak naopak chceme, aby při držení klávesy hráč plynule chodil. Nyní by hra měla vypadat takto:

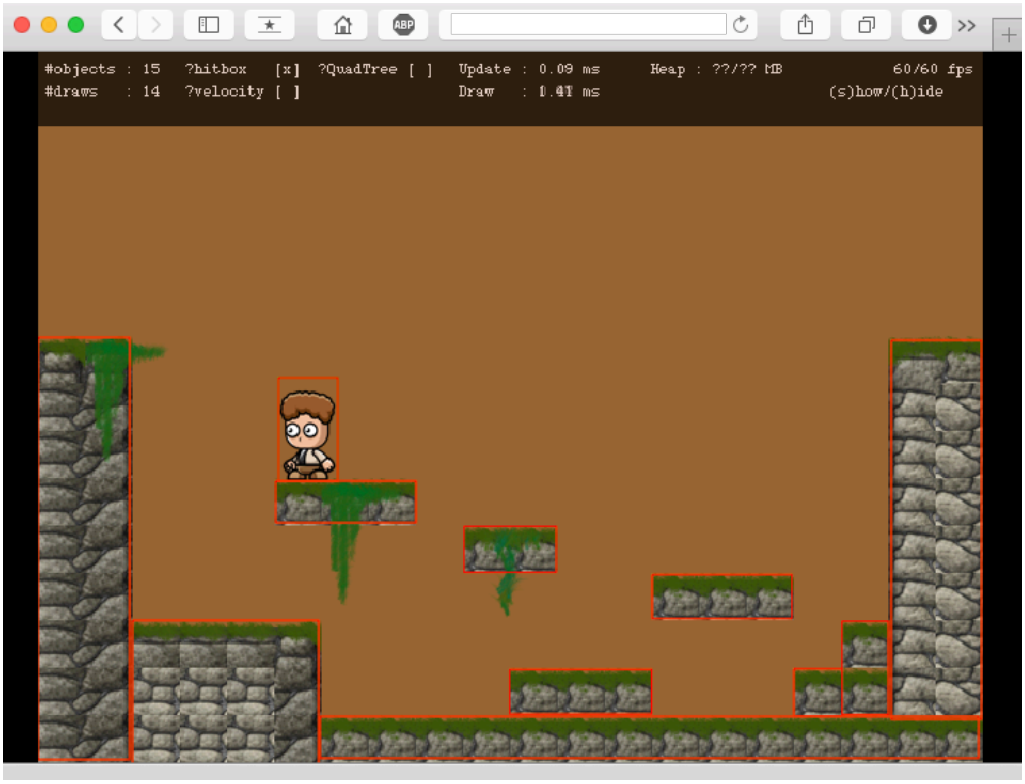


**Obrázek 11 –** Hra s chodící postavou hráče

## 7.2 Testovací plugin

V základním balíčku boilerplate MelonuJS je i debugovací (testovací) plugin, který lze zapnout tak, že do lišty s URL adresou v prohlížeči připíšeme „#debug“ a aktualizujeme stránku. Čímž se spustí tento plugin a krásně nám zobrazí hrany

vykreslených objektů. Toto může být velice užitečný nástroj při odhalování chyb při vývoji hry.



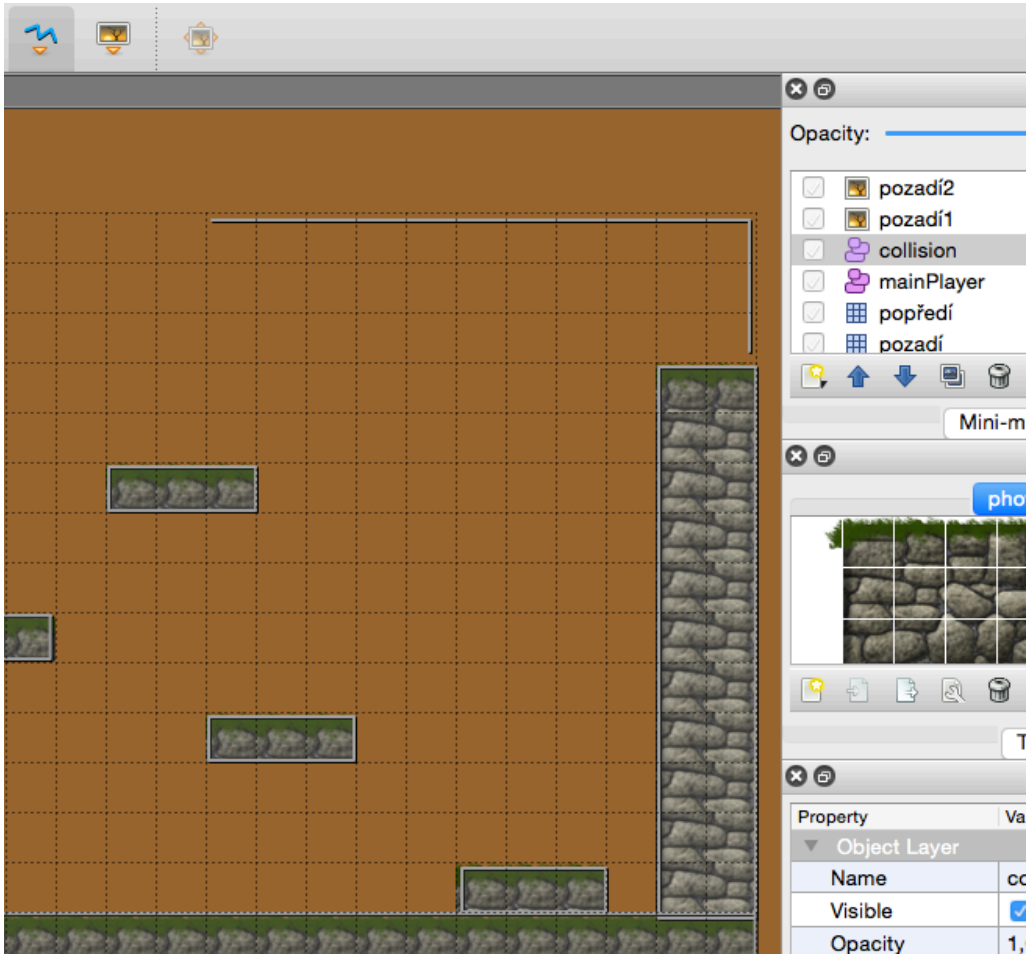

**Obrázek 12 –** Testovací plugin

# 8 Parallax pozadí

Parallax efekt zajistí, že hra vypadá více prostorově. K tomu, abychom docílili tohoto efektu, je třeba mít minimálně dva obrázky. Jeden může být klasický JPG obrázek. Ten druhý, který bude nad tím prvním obrázkem, musí být záměrně ve formátu PNG. Využijeme tak transparentnosti PNG obrázku, aby skrz něj byl vidět ten v pozadí.

Dále je třeba rozpohybovat mapu, toho docílíme tak, že si zvětšíme naši mapu do šířky. To uděláme přes nástroj „Map – Resize map“. Výšku (height) necháme na 15ti dlaždicích a šířku (width) zvětšíme z 20ti na 45 dlaždic. Až se mapa roztáhne, tak si musíme poupravit prostředí podobně jako na začátku. Doplníme dlaždice do vrstvy „pozadí“ i „popředí“ a dokreslíme kolizní objekty podle nově vzniklých dlaždic (collision). Nyní v kolizní vrstvě přiděláme i objekty okolo mapy tam, kde by se mohlo stát, že hráč vypadne z obrazovky pryč. Mohou to být opět klasické objekty (čtverec, obdélník), ale i úsečky.

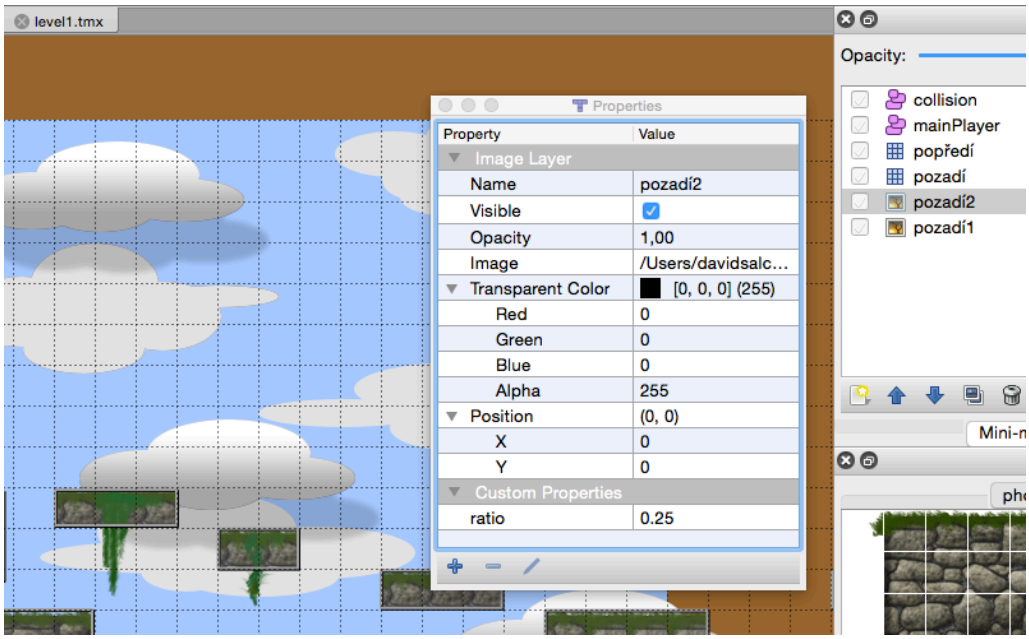
Poté vytvoříme nové vrstvy pro obrázky (Image layer), pozadí1, pozadí2 a pozadí3.




**Obrázek 13 –** Vrstvy pro obrázky

Pravým tlačítkem rozbalíme možnosti vrstvy (Layer properities) a v položce foto (image) vybereme obrázky mraky.png, mrakyBig.png a grass.png. Prvnímu obrázku přidáme nový atribut „ratio“ s hodnotou 0.35. Druhému objektu „pozadí2“ přidáme též atributt „ratio“, kterému nastavíme hodnotu 0.25. Poslednímu obrázku nastavíme „ratio“ na 0.55. To znamená, že se bude posouvat s poloviční rychlostí oproti posunu mapy. Pokud bychom prvnímu obrázku nenastavili žádný atribut, pak by zůstal statický. Čím je hodnota „ratio“ menší, tím je posouvání obrázků pomalejší. Maximální hodnota je pak 1, obrázek se pak posouvá 1:1 s pohybem mapy.

Přestože jsou obrázky jinak veliké, Tiled je bude opakovat po ose X i Y a tak vyplní všechen prostor mapy po spuštění hry. Zároveň tyto obrázkové vrstvy posuneme až na poslední místa, aby nepřekrývaly ostatní vrstvy, jako jsou dlaždice, hráč apod.




**Obrázek 14** – Vložení obrázků a nastavení parallax efektu

Na závěr opět přiřadíme obrázky do resources.js, aby se mohly načíst a my je vyzkoušet.

```
{name: "mraky",type:"image",src: "data/img/mraky.png"},

{name: "mrakyBig",type:"image", src:
"data/img/mrakyBig.png"},

{name: "grass",          type:"image", src:
"data/img/grass.png"},
```

Takto by měla vypadat hra nyní. Když se projdeme s hráčem po mapě, tak se parallax efekt, krásně projeví.

**Obrázek 15 –** Parallax pozadí ve hře

# 9 Sbírání předmětů

K principu 2D her patří sbírání nějakých předmětu ku prospěchu hráče. Nyní si tedy přidáme několik předmětů, které budeme moci sbírat (CollectableEntity). Bude se jednat o předměty s různým bodovým ohodnocením, které nám přidají kredity (skóre) a nápoje či lektvary (potions) pro obnovení pokusů (života) a rychlosti.

Tyto předměty mohou být i statické, čili jeden PNG obrázek o velikosti kupříkladu 30x30 pixelů. Zároveň je zde stejný princip animování jako u předešlých objektů, tedy animování pomocí šířky jednoho okénka v obrázku (spritewidth).

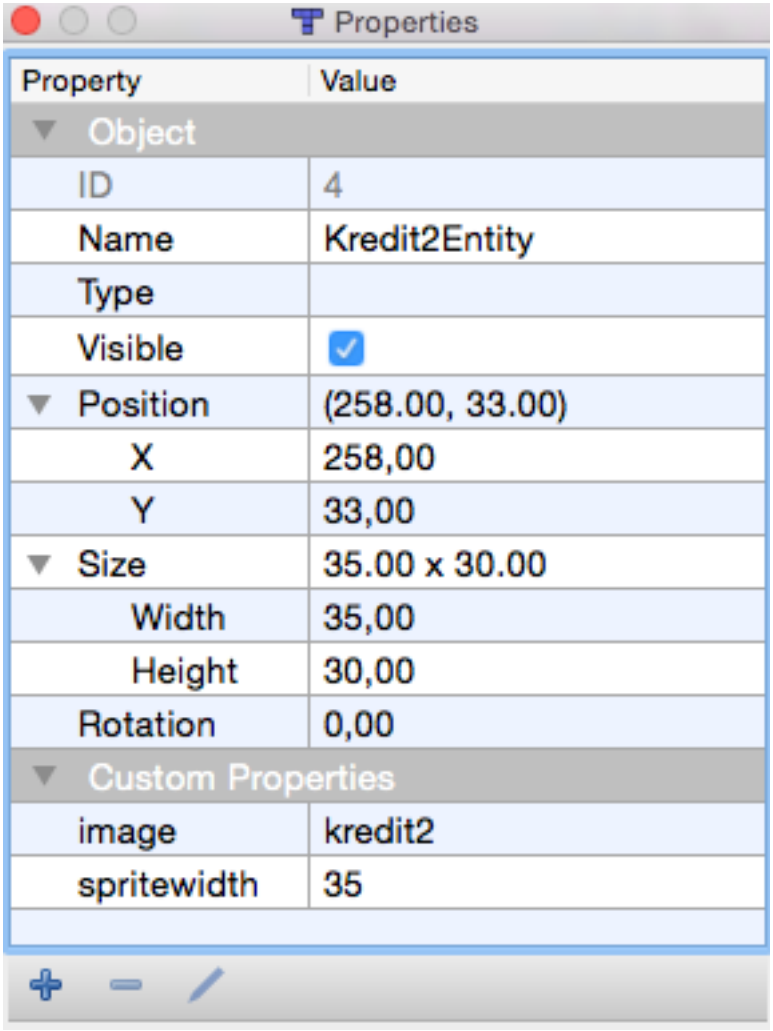
K tomuto účelu použijeme tyto obrázky, které si rovnou napíšeme do resources.js.

```
{name:        "kredit2",                type:"image",src:
"data/img/sprite/kredit2.png"},

{name:        "kredit4",                type:"image",src:
"data/img/sprite/kredit4.png"},

{name:        "kredit6",                type:"image",src:
"data/img/sprite/kredit6.png"},
```

Později budeme potřebovat i zobrazení počtu kreditů (skóre) na obrazovce a k tomu slouží buď jednořádkový bitmapový font:

```
{name:"32x32font",type:"image",src:
"data/img/font/32x32font.png"},
```

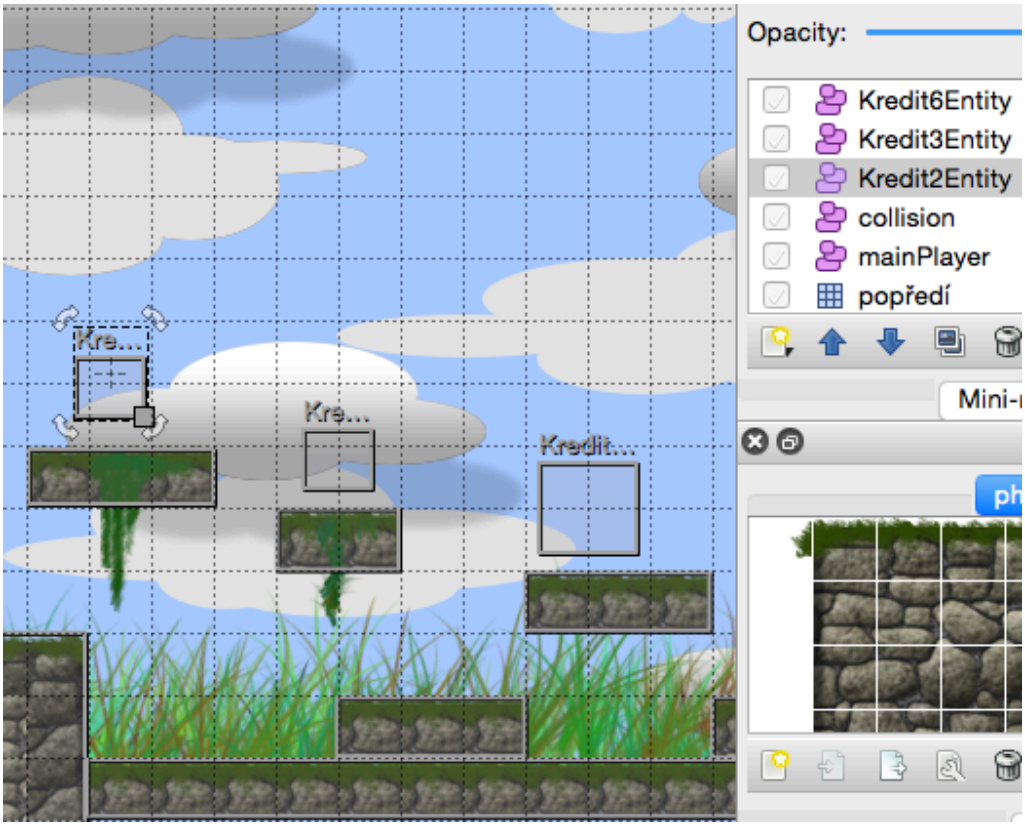
A nebo vykreslení systémového fontu s pomocí objektu `me.Font` a funkce `draw()`, k tomu ale později. Začneme tedy s tím, že v Tiled vytvoříme 3 objektové vrstvy pro obrázky kreditů. Vrstvy pojmenujeme tak, jak se budou jmenovat v entities.js, aby je MelonJS později mohl správně nahrát do mapy, zde tedy Kredit2Entity, Kredit3Entity a Kredit6Entity.




**Obrázek 16 –** Nastavení předmětu pro sbírání

Toto je nastavení prvního předmětu Kredit2Entity. Přidané atributy „image“ a „spritewidth“ určují o jaký obrázek se jedná a jak je široký. Zároveň zde musí být nastavena velikost (size), šířka 35 a výška 30 (pixelů).

Další předměty, Kredit3Entity a Kredit6Entity, přidáme stejným způsobem jen u Kredit6Entity pozměníme hodnoty výšky, šířky a hodnoty spritewidth. (50x46 pixelů).




**Obrázek 17** – Objekty pro sbírání

Nyní přidáme konstruktor těchto objektů do entities.js. Protože se jedná o objekt, který budeme sbírat, tak jej vytvoříme z třídy me.CollectableEntity, která slouží pro účely těchto předmětů.

```
game.Kredit2Entity = me.CollectableEntity.extend({
  init: function (x, y, settings){
  this._super(me.CollectableEntity, 'init', [x, y , settings]);},
    onCollision : function (response, other) {
        switch(other.body.collisionType){
            case me.collision.types.PLAYER_OBJECT:
                game.data.score += 2;

this.body.setCollisionMask(me.collision.types.NO_OBJECT);
```

```
  me.game.world.removeChild(this);
   return false;
break;
case me.collision.types.ENEMY_OBJECT:
     return false;
     break;
}
}
});
```

Tyto konstruktory budeme mít tři, pro každý objekt zvlášť. Už jen z důvodu různé velikosti toho posledního. Zde se budou lišit pouze v počtu přidaných bodů. A to ovlivňujeme tak, že měníme proměnnou `score`, která byla definována na začátku v game.js a to tímto způsobem:

```
game.data.score += 2;
```

Podobně jako u objektu hráče, tak i zde má objekt inicializační funkci (init) a funkci, která je volána při kolizi s ostatními objekty (onCollision). Protože se může stát, že později při tvorbě hry se nepřátelé dotknou tohoto objektu, tak zde použijeme přepínač (switch), abychom rozlišili chování podle typu objektu.

```
switch(other.body.collisionType){
case me.collision.types.PLAYER_OBJECT:
```

MelonJS rozlišuje několik kolizních typů, které můžeme jednotlivým objektům nastavit. Zde v kolizní funkci (onCollision) kreditu kontrolujeme, zda se ho nedotýká objekt s typem PLAYER_OBJECT. Pokud ano, pak přičteme kredity do globální proměnné a odstraníme tento objekt. Objekt odstraníme spolehlivě tak, že nejdříve nastavíme objektu typ NO_OBJECT, aby v ten moment, přestal reagovat na okolí a ihned ho odebereme z herního světa pomocí funkce `removeChild()` s atributem `this`, čímž ukážeme na tento konkrétní právě sebraný předmět. :

```
this.body.setCollisionMask
(me.collision.types.NO_OBJECT);
me.game.world.removeChild(this);
```

V poslední řadě musíme ještě přidat tyto objekty do herního světa při načítání hry v game.js.

```
me.pool.register("Kredit2Entity", game.Kredit2Entity);
me.pool.register("Kredit3Entity", game.Kredit3Entity);
me.pool.register("Kredit6Entity", game.Kredit6Entity);
```

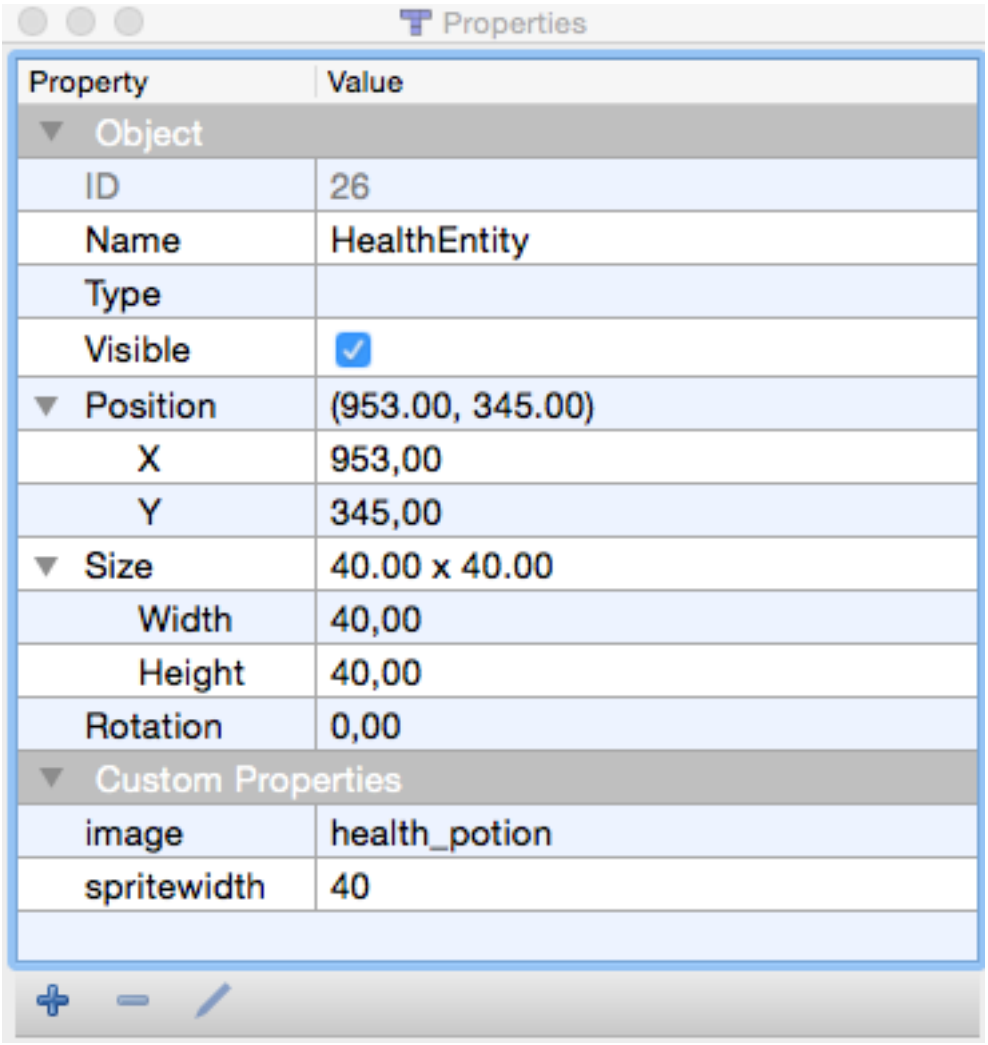
A nyní hra vypadá takto:

**Obrázek 18** – Objekty kreditů ve hře

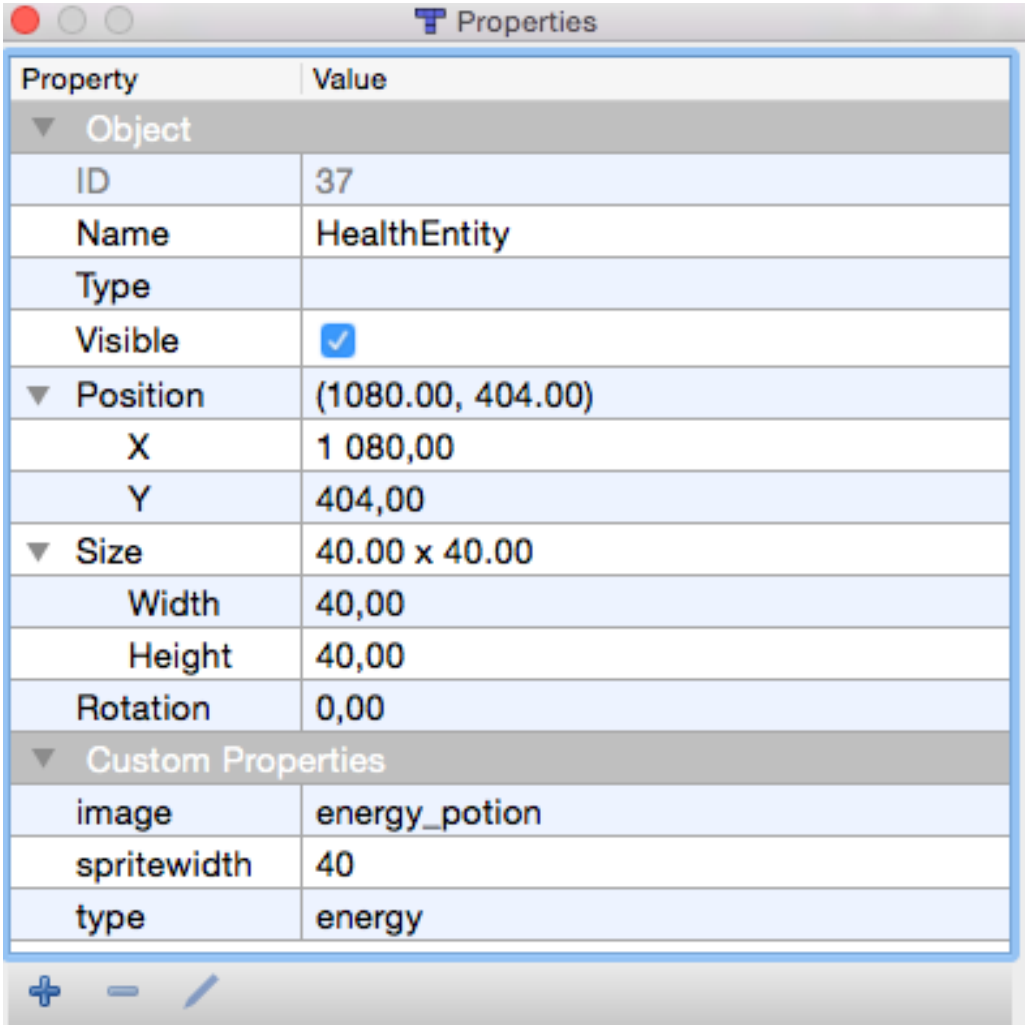
## 9.1 Život a energie

Dalším objektem pro sbírání bude lektvar k doplnění života (health potion) a lektvar, který hráče zrychlí (energy potion). Zde si vyzkoušíme, jak funguje jeden konstruktor společně s různým nastavením objektu v Tiled editoru.

Samotný konstruktor objektu v entities.js je stejný jako u předchozích objektů, které jdou sebrat. Inicializační funkce (init) a funkce řešící kolize (onCollision) je v základu stejná. Pouze v nastavení objektu v Tiled (object properities), přidáme atribut „type“ pro rozlišení předmětů.




**Obrázek 19 –** Konstruktor pro objekt HealthEntity




**Obrázek 20 –** Konstruktor pro objekt EnergyEntity

Protože budeme používat pouze jeden konstruktor v entities.js, je důležité, aby se i tyto dva objekty jmenovaly stejně „HealthEntity“. Dále je nutné, aby obrázky pro tyto dva objekty byly stejné, jinak by se nemohly vytvářet z jednoho konstruktoru. Pro oba je zde stejný atribut „spritewidth“ nastaven na 40 (pixelů). Tyto objekty se liší v přiřazeném obrázku energy_potion.png a health_potion.png. Pokud se má energy_potion chovat jako rozdílný objekt, je nutné ho nějakým způsobem odlišit. Proto přidáváme atribut „type“ s hodnotou kupříkladu „energy“. Později lze přidat různé množství dalších atributů pro zpestření hry.

Do inicializační funkce (init), přidáme tento kód:

```
var type = settings.type;
```

Tímto jsme si založili lokální proměnnou `type`. Abychom získali hodnotu atributu z Tiled objektu musíme volat `settings` což je nastavení Tiled objektu (properities) a poté jméno atributu, který žádáme. Vytvoříme si globální proměnnou v game.js se jménem `walking` (chození), abychom ji mohli měnit z těla funkce `onCollision()`. v tělu kolizní funkce objektu HealthEntity přidáme tento kód:

```
if(game.data.life <6){
                    game.data.life += 1;
 }
if(this.type == "energy"){
                    game.data.walking +=4;
                    me.timer.setTimeout(function () {
game.data.walking -=4; },7000, true);
}
```

Pokud sebereme objekt HealthEntity a máme méně než 6 pokusů či života (game.data.life), pak tuto proměnnou zvětšíme o 1. Toto se provede vždy. Dále kontrolujeme zda výše vytvořená proměnná má hodnotu „energy“. u lektvarů, kterým nenastavíme v Tiled editoru žádný atribut „type“ bude tato hodnota rovna `null`. Pokud se ale proměnná nastaví na hodnotu „energy“, pak se globální proměnná `walking` zvýší o 4. Dále se provede funkce `setTimeout()`, která má parametry funkce, čas a zda ji lze pozastavit. Zde máme nastavenou funkci tak, aby se hodnota `walking` opět snížila o 4 a to po 7000 milisekundách. Poslední hodnota true nám říká, že funkce se pozastaví, pokud přepneme na jiné okno prohlížeče apod.

Pokud je objekt sebrán, pak ho odebereme z herního světa přesně stejným způsobem jako předchozí objekty. Protože se jedná o nový objekt, nesmíme jej zapomenout přiřadit do herního světa v game.js.

```
me.pool.register("HealthEntity", game.HealthEntity);
```

### 9.1.1 Aktualizace setVelocity()

Protože jsme si vytvořili globální proměnnou `walking` a při sebrání objektu HealthEntity s atributem energy tuto proměnnou zvětšujeme, musíme poupravit kód hráče, aby se tyto změny aplikovaly na chování postavy hráče.

Základní nastavení rychlosti chození a velikosti skákání hráče je optimálně asi takovéto `this.body.setVelocity(6,15)`. Pokud ale chceme, aby se tato hodnota měnila, musíme místo prvního atributu přiřadit proměnnou `walking`. Protože hodnota této proměnné se změní za krátký čas, je třeba kontrolovat hodnotu této proměnné. Pokud bychom tak neučinili funkce `setTimeout()` by sice změnila hodnotu proměnné zpět na původní hodnotu, ale postava hráče by byla stále rychlá jako při jejím prvním zvýšení.

Proto tuto hodnotu budeme kontrolovat v objektu PlayerEntity v těle funkce `update()`. Víme, že základní hodnota chození je 6, tak si vytvoříme lokální proměnnou `walk` o stejné velikosti. A s touto proměnnou poté porovnáváme hodnotu globální proměnné `walking`. Pokud nejsou stejné, pak nastavíme hráčovi rychlost právě na globální proměnnou. Pokud již nejsou stejné, pak přenastavíme rychlost zpět na původní velikost, tedy hodnotu `walk`.

```
var walk = 6;
        if(walk != game.data.walking){
this.body.setVelocity(game.data.walking,15);}
        else{ this.body.setVelocity(walk,15);}
```

# 10 Nepřátelé a překážky

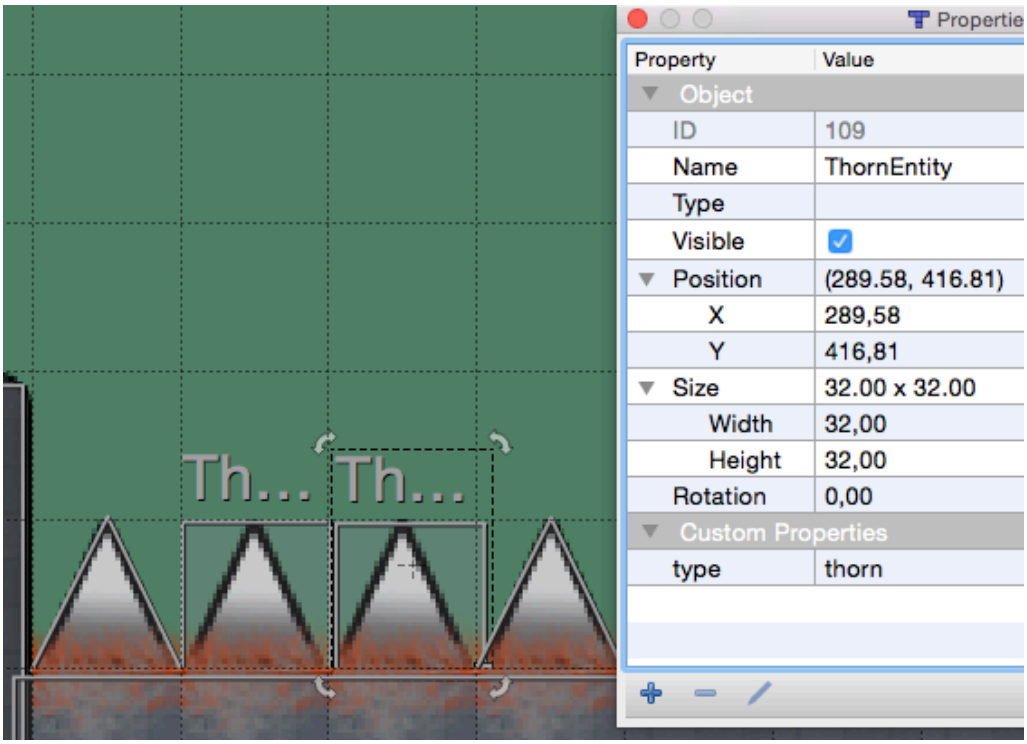
Pro vytvoření nepřátel máme připravené obrázky angry_enemy.png a flying_enemy.png. a jednu překážku bez obrázku. Vytvoříme si dva typy nepřátel, jeden chodící a druhý létající ve vzduchu. Přidáme si tyto obrázky do resources.js.

```
{name: "angry_enemy",       type:"image",   src:
"data/img/sprite/angry_enemy.png"},

{name: "flying_enemy",       type:"image",  src:
"data/img/sprite/flying_enemy.png"},
```

Dále v entities.js definujeme jako nový objekt EnemyEntity a FlyingEntity podobně jako u hráče.

```
game.EnemyEntity = me.Entity.extend({ });
```

Všechny tyto objekty, ať už nepřátelské či předměty k sebrání, definujeme též jako náležitě pojmenované objekty v Tiled editoru s šířkou, výškou, obrázkem a dalšími atributy. Později se dostaneme i k vytváření jiného typu nepřátelského objektu a tak bych zde rád podotknul, že lze objekty tvořit samozřejmě i bez obrázků a tedy i přesného konkrétního rozměru. Později budeme vytvářet jiné typ nepřátelského objektu (ThornEntity) a ten lze vytvořit v Tiled editoru i tak, že se obkreslí terén, podobně jako se vytváří kolize s mapou a místo toho, aby se nechal nevyplněný a prázdný, tak jej pojmenujeme jako ThornEntity, či jinak, a dle libosti přidáváme atributy.




**Obrázek 21 –** Dlaždice (tileset) jako jiný objekt

Na obrázku můžete tedy vidět, že jde o nakreslený objekt hrotu, který je součástí tilesetu (dlaždic). Tento objekt lze tedy obkreslit, jak pomocí „Rectangle tool“, tak „Polygon tool“ pro přesnější vystižení tvaru tohoto objektu. Podle jména objektu poté vytváříme pouze jednoduchý konstruktor v entities.js, ve kterém nastavíme typ objektu a kolizní funkci s přepínačem tak, aby ignoroval základní kolizi s `ENEMY_OBJECT` a `PLAYER_OBJECT`.

```
game.ThornEntity = me.Entity.extend({
    init: function (x, y, settings){
        this._super(me.Entity, 'init', [x, y , settings]);
        this.body.collisionType= me.collision.types.ENEMY_OBJECT;
    },
    onCollision : function (response, other) {
        switch(other.body.collisionType){
```

```
            case me.collision.types.PLAYER_OBJECT:
                return false;
                break;
         case me.collision.types.ENEMY_OBJECT:
                return false;
                break;
        }
    }
});
```

Zde zajistíme pouze vytvoření tohoto objektu, ale v základu jím může procházet nepřátelský objekt i hráč. Kolizi s tímto objektem řešíme později.

## 10.1 Funkce init()

Zde bude kód opět o něco delší. Začneme tedy s inicializační funkcí (init), která bude vypadat takto:

```
init: function (x, y, settings){
        settings.image = "angry_enemy";
        var width = settings.width;
        var height = settings.height;;
        settings.spritewidth = settings.width = 64;
        this._super(me.Entity,    'init',    [x,    y    ,
settings]);
        x = this.pos.x;
        this.startX = x;
        this.endX = x + width - settings.spritewidth
        this.pos.x= x + width - settings.spritewidth;
        this.updateBounds();
```

```
        this.walkLeft = false;
        this.body.setVelocity(1, 6);
        this.alwaysUpdate = true;
    },
```

Tento kód je stejný pro oba objekty jak EnemyEntity tak FlyingEntity. v této inicializační funkci se liší opět jen v nastavení šířky objektu a v přiřazeném obrázku:

```
settings.image = "angry_enemy";
settings.spritewidth = settings.width = 64;
```

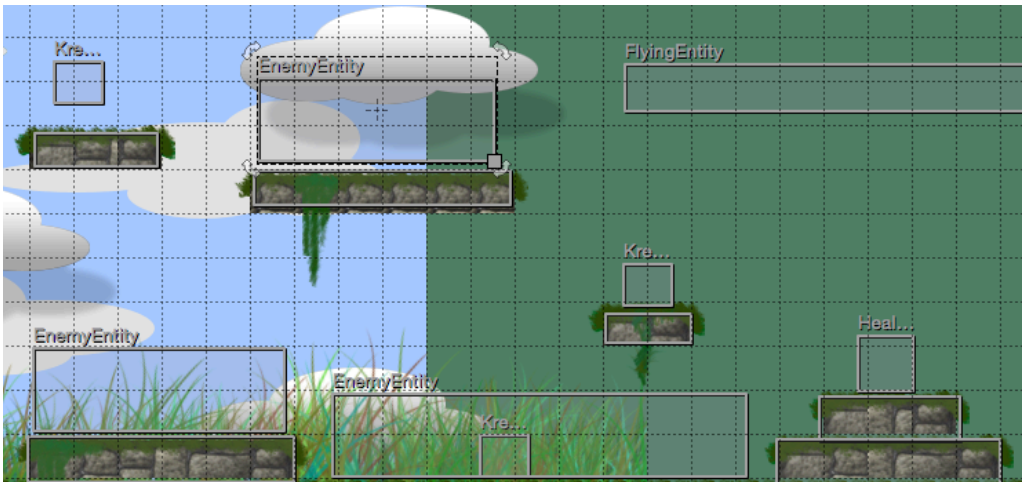
Protože i tyto objekty mají své nastavení `setVelocity()`, lze si opět přednastavit pomocí Tiled editoru a lokální proměnné, aby každý objekt čili nepřítel chodil různou rychlostí.

Pokud s tímto nastavením vytvoříme i objekt FlyingEntity pak sice bude fungovat, ale nebude držet ve vzduchu. Pro letající objekt tedy speciálně v této funkci definujeme gravitaci a nastavíme jí na hodnotu 0. Pro tento objekt tedy vypneme gravitaci a tak bude držet na místě, kam jsme jej umístili v Tiled.

```
this.body.gravity = 0;
```

## 10.2 Funkce update()

V této funkci se postaráme o pohyb nepřátel. Zde se odkazujeme na proměnnou `walkLeft` či `flyLeft`, která je nastavena na hodnotu true. Touto proměnnou měníme a kontrolujeme směr chození či létání. v inicializační funkci jsme si definovali také proměnné `x`, `startX`, `endX` a `pos.x`, které nám udávají rozměry objektu podle Tiled editoru. Podle šířky objektu v Tiledu se vypočítá trasa po, které bude nepřítel chodit.




**Obrázek 22** – Objekty nepřátel v Tiled editoru

Takto by měla vypadat funkce `update()`:

```
update : function (dt){

if (this.alive){

    if (this.walkLeft && this.pos.x <= this.startX){

        this.walkLeft = false;

     }

     else if (!this.walkLeft && this.pos.x >= this.endX){

      this.walkLeft = true;

     }

      this.renderable.flipX(this.walkLeft);

      this.body.vel.x += (this.walkLeft) ? -
this.body.accel.x * me.timer.tick : this.body.accel.x *
me.timer.tick;

}



else{

    this.body.vel.x = 0;

}
```

```
this.body.update(dt);

me.collision.check(this);

return (this._super(me.Entity, 'update', [dt]) ||
this.body.vel.x !== 0 || this.body.vel.y !== 0);

},
```

Zbývající funkci `onCollision()` najdeme níže.

# 11 Funkce onCollision()

Nyní již máme ve hře dostatek objektů a tak se bude muset rozvádět funkce `onCollision()` trochu více.

## 11.1 Funkce onCollision() u nepřátel

Protože ve hře je několik typů objektů, je výhodné řešit kolizi přes přepínač (switch). U nepřátelského objektu EnemyEntity, který je nastaven na typ `ENEMY_OBJECT`, řešíme kolizi s okolním světem a to z důvodu, aby byl schopný chodit po terénu a nepropadnout. Dále kolizi s hráčem, který je typu `PLAYER_OBJECT`. Zároveň musíme řešit kolizi, pokud se objekt střetne s objektem stejného typu, aby se mohlo více nepřátel pohybovat mezi sebou, aniž by se ovlivnili a naráželi do sebe. A v poslední řadě typ `COLLECTABLE_OBJECT`, což jsou všechny předměty, které lze sebrat. Bez tohoto nastavení by se pak stalo, že pokud by v cestě objektu byl objekt, který se dá sebrat, tak by jej nepřítel sebral, což samozřejmě nechceme. Takto by měl vypadat výsledný kód kolize u nepřátelského objektu EnemyEntity:

```
onCollision : function (response, other) {
 switch(other.body.collisionType){


   case me.collision.types.PLAYER_OBJECT:
       if(this.alive&&  (response.overlapV.y  >  0)  &&
response.a.body.falling) {
         if (this.renderable.isFlickering()){
             this.alive = false;
             var self = this;
               me.game.world.removeChild(self);
         }
       else{
           this.alive = false;
           var self = this;
```

```
this.body.setCollisionMask(me.collision.types.WORLD_SHAPE
);
this.renderable.flicker(300,function(){me.game.world.remo
veChild(self)});
        }
        game.data.score+=5;
     }
    if (this.alive && (response.overlapV.y <= 0) ||
response.a.body.falling || response.a.body.jumping) {
     this.renderable.flicker(300,function(){
     game.data.score-=1;});
     }
     return false;
break;
case me.collision.types.COLLECTABLE_OBJECT:
     return false;
break;
case me.collision.types.ENEMY_OBJECT:
     return false;
break;
}
return true;
}
```

Pokud tedy dojde ke kolizi s hráčem, řeší se dvě podmínky. Zda je objekt aktivní či v přeneseném významu živý, díky pomocné proměnné `this.alive`. Pokud je tato hodnota `true` pak, musíme zjistit, jakým způsobem došlo ke kolizi. Zde definujeme, jakým způsobem může hráč zneškodnit nepřátelský objekt. Pokud tedy hráč prošel objektem, či proskočil, pak nepřátelský objekt rozblikáme na 300 milisekund

a snížíme hráčův počet kreditů z globální proměnné `score`. Klíčové zjištění překryvu objektu řešíme na ose Y a to pomocí `response.overlapV.y <= 0`. Dále bereme v potaz klíčové proměnné od hráče, zda byl ve skoku či zrovna padal dolů to zjistíme pomocí `response.a.body.falling` a `response.a.body.jumping`.

V opačném případě řešíme, zda se hráč dotkl objektu bez postranního překryvu a byl ve stavu padání. Zde tedy změníme znaménko překryvu na `response.overlapV.y > 0` a kontrolujeme, aby stav padání byl `true`, `response.a.body.falling`. Pokud se tak stane rozblikáme nepřátelský objekt na 300 milisekund, zvětšíme globální proměnnou `score` a nepřátelský objekt odebereme ze světa pomocí `me.game.world.removeChild(self)`.

Daleko jednodušší bude řešení kolize s objekty stejného typu, tedy `ENEMY_OBJECT` a typu `COLLECTABLE_OBJECT`. Zde pouze vrátíme `false`, čímž vyjádříme, aby v případě kolize na tento objekt nereagoval.

Na konci funkce `onCollision()` poté vrátíme `true` a to z důvodu, aby vše ostatní čili i okolní svět, byl pevný a objekt nepropadával.

## 11.2 Funkce onCollision() u hráče

V herním světě již máme spoustu objektů a nyní se to promítne i do zatím prázdné funkce onCollision(). Zde toho budeme řešit více, protože objekt hráče je samozřejmě více využívaný a je v interakci s okolím daleko častěji než ostatní objekty. Samotný návrh řešení kolize u hráče a volání funkcí může mít zde spoustu řešení a záleží čistě na programátorovi, jak si danou věc rozmyslí. Já jsem šel touto cestou. Vše samozřejmě opět, řešíme přes přepínač (switch) kterému zavedeme defaultní stav, jenž bude vracet `false`. a na konci celé funkce vracíme `true`, aby objekt sám o sobě reagoval a choval se jako pevný.

```
default: return false;
```

Bez tohoto defaultního stavu, by sice vše do jisté míry fungovalo, ale mohlo by se stát, že objekt hráče by se nějakým způsobem překryl kupříkladu s nepřátelským objektem v době problikávání a dostal by se do opravdové kolize dvou objektů.

Objekty by mezi sebou problikávaly, náhodně skákaly, či by vypadly z prostředí apod.

### 11.2.1 Kolize s typem ENEMY_OBJECT

Pokud se objekt hráče dostane do kolize s nepřátelským objektem, jsou zde tři možnosti, jak může situace dopadnout. Podobně jako v definici kolize u nepřátelského objektu zde řešíme, zda hráč skočil právě a přímo na tento objekt. Toto je řešeno opět metodou překryvu `response.overlapV.y>0` a stavem padání hráče `this.body.falling`. Pokud se tak stane, vyvoláme umělý skok, aby se hráč odrazil od objektu. Tohoto plynulého skoku docílíme rychlého vertikálního zrychlení nastaveného na maximální velikost z velikosti pro skok hráče, v tomto případě v základu nastavenou na hodnotu 15 (setVelocity), tedy `this.body.vel.y = -this.body.maxVel.y * me.timer.tick`. a gravitace se postará o zpomalení a vrácení hráče zpět na zem. Zde musíme nastavit hráčův stav na skákání `this.body.jumping` na `true`.

Dále jako efekt můžeme využít otřesu obrazovky v obou osách X i Y a to zajímavou funkcí `me.game.viewport.shake()`.

Toto se stane, pokud jsme řádně skočili na nepřátelský objekt. v opačném případě budeme řešit obdobnou situaci jako u všech 2D skákacích her. Pokud se dotkneme nepřátelského objektu a máme více než jeden pokus či život, tak jej ubereme z globální proměnné game.data.life-=1 a pomocí funkce flicker() jej rozblikáme hráče na 5000 milisekund.

Může nastat i případ, že se znovu dotkneme nepřátelského objektu, ale jsme ve stavu, kdy blikáme. Tento stav blikání obecně značí, že je hráčova postava po krátkou dobu nezranitelná, aby se dostal včas do bezpečí. V tomto případě musíme do podmínek přidat funkci `isFlickering()`, která vrací `true` nebo `false` podle toho, zda postava hráče bliká. Díky této funkci tedy můžeme zajistit, že dokud hráč problikává, tak na něj nebude platit odebírání života apod. Pokud tedy hráč již nebliká a globální

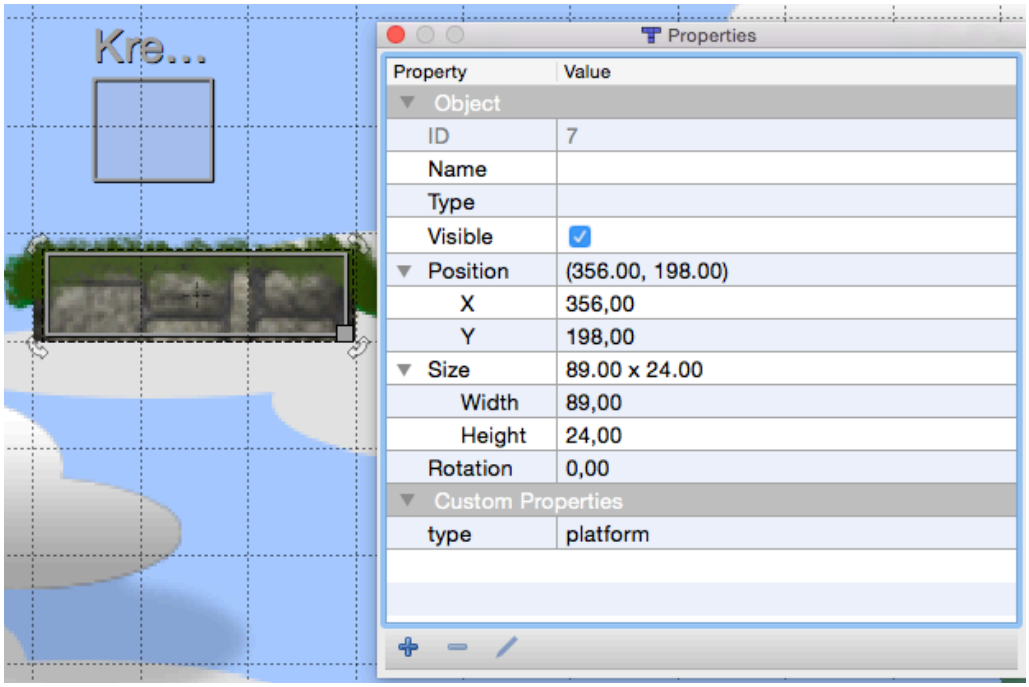
proměnná pro život či pokus je rovna 1, pak ji nastavíme na hodnotu 0, odebereme hráče z herního světa.

```
if ((response.overlapV.y>0) && this.body.falling) {
     this.body.vel.y    =    -    this.body.maxVel.y    *
me.timer.tick;
this.body.jumping = true;

me.game.viewport.shake(10,500,me.game.viewport.AXIS.BOTH)
;
 }
else {
     if(!this.renderable.isFlickering()&&
     game.data.life>1){
     game.data.life-=1;
     this.renderable.flicker(5000);
     }
     if(!this.renderable.isFlickering()                  &&
     game.data.life==1){
     game.data.life=0;
     me.game.world.removeChild(this);
}
}
```

### 11.2.2 Zavedení objektů typu PLATFORM

Platformy jsou vlastně plošinky, pevné objekty terénu, které visí ve vzduchu a lze jimi proskočit zezdola směrem nahoru, ale v opačném směru jsou již pevné. Toto řešil MelonJS elegantně v předchozích verzích s meta tileset dlaždicemi a automatickou funkcí `onPlatform()`. v nových verzích již tato možnost není a tak musíme toto dělat ručně. Abychom tento princip zavedli, musíme zpět do Tiled

editoru do vrstvy „collision“ a těm objektům, které chceme, aby se chovaly jako plošiny, platformy, nastavíme nový atribut „type“ na hodnotu „platform“.




**Obrázek 23** – Nastavení objektu plaftorm v Tiled editoru

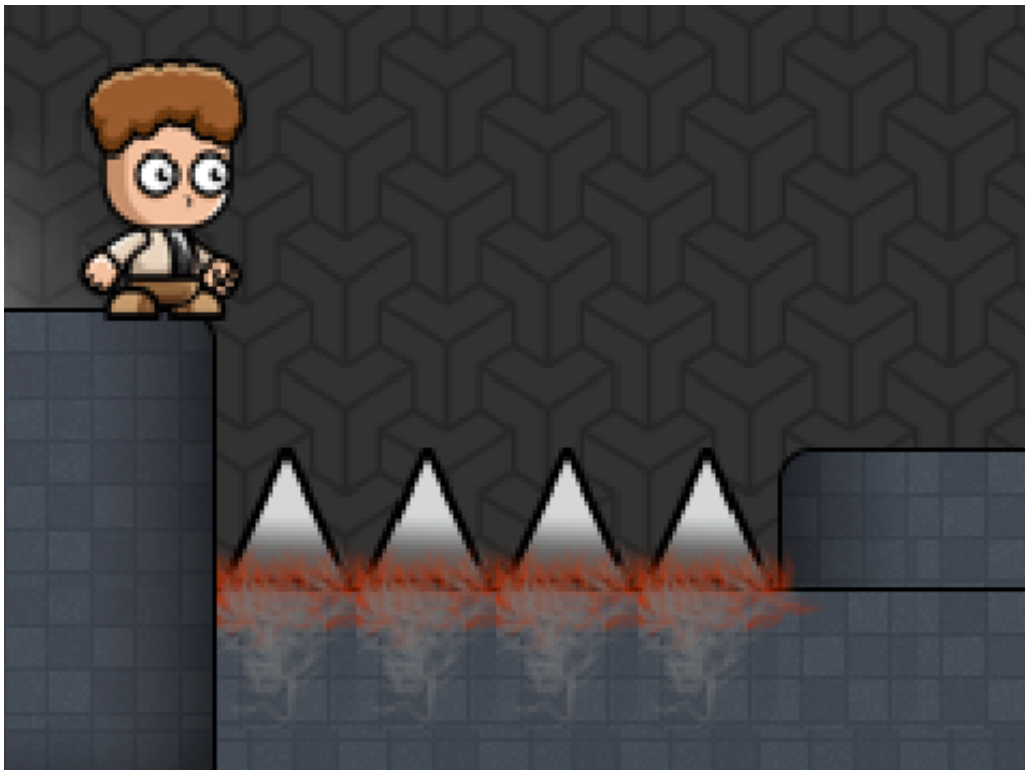
Na konci funkce `onCollision()` vždy vracíme `true` což znamená, že ke všem objektům se chováme jako k pevným, zejména k okolnímu světu a ten je typu `WORLD_SHAPE`. a protože nyní jsme editovali v Tiled editoru klasický objekt, který by za normálních okolností byl pevný, tak jej musíme odfiltrovat od ostatních a pro tento typ nastavit nové chování postavy hráče pokud je s ním v kontaktu.

Zde budeme kontrolovat, zda objekt nemá náhodou definovaný atribut „type“ a to přes `other.type === „platform“`. A pokud ano, pak kontrolujeme zda je postava hráče ve stavu padání, není stisknutá klávesa dolů a opět kontrolujeme překryv na ose Y. Pokud toto vše je splněno vrátíme `true` a na plošince bude možno stát a proskočit jí zezdola. Musíme ale dávat pozor na nastavení překryvu na ose X, který musíme nastavit na hodnotu 0. Toto nastavení je zde z důvodu, aby postava hráče mohla procházet touto plošinkou, pokud bude kupříkladu nízko. Jinak bychom proskočili na plošinku přesně kolmo zezdola či skočili přímo na plošinku, ale ze strany bychom se zasekli. Pokud výše zmíněné podmínky neplatí, pak vracíme `false`. To pak bude mít za následek nereagování na plošinku, pokud hráč nedoskočí na plošinku, čili nevyskočí tak vysoko.

```
case me.collision.types.WORLD_SHAPE:
   if (other.type === "platform") {
     if(this.body.falling&&
!me.input.isKeyPressed('down') &&
 (response.overlapV.y  >  0)  &&  (~~this.body.vel.y  >=
~~response.overlapV.y)) {
                response.overlapV.x = 0;
                return true;
     }
return false;
    }
break;
```

### 11.2.3 Rozlišení dvou typů ENEMY_OBJECT

Framework sice nabízí několik základních typů objektů, ale při delší tvorbě hry nemusí stačit a budeme potřebovat nějakým způsobem ještě detailněji odlišit nepřátelské objekty. Ve hře máme létající a chodící nepřátele, kteří mají stejný princip zneškodnění, čili skok na hlavu tohoto nepřátelského objektu. Dále jsem do hry přidal objekt hrotu či bodáku (thorn) a chtěl jsem odlišit chování kolize s tímto objektem. Protože již pro jednotlivé typy objektů máme nějaké nastavení kolize v momentě, kdy nový objekt nastavíte na určitý typ objektu, začnou se na něj samozřejmě vztahovat pravidla pro něj určená.



**Obrázek 24 –** Objekt hrotu (thorn)

Na obrázku můžete vidět, jak tento objekt vypadá a jeho samotný vzhled napoví, že by na něj hráč neměl stoupat. Svojí logikou ale patří pod nepřátelský objekt `ENEMY_OBJECT`. Pokud bychom jej tedy nechali s prostým nastavením typu objektu, po skoku na hrot by objekt zmizel a hráčovi by se přičetly body, což nechceme.

Protože MelonJS je hlavně založen na spolupráci s Tiled editorem, budeme muset začít s rozlišením typu objektu již v editoru. Do nastavení objektu přidáme atribut „type“ a hodnotu můžeme nastavit třeba právě „thorn“. Dále všem dosavadním nepřátelským objektům též musíme přidat atribut „type“ a nastavit jej na nějakou rozlišovací hodnotu, v mém případě jsem nastavil hodnotu „enemy“.

V hráčově kolizní funkci máme prozatím nastavenou kolizi pro klasický `ENEMY_OBJECT` a ostatní. Dosavadní kód pro nepřátelský objekt ponecháme nedotčený, pouze vnitřek `case me.collision.types.ENEMY_OBJECT:` rozdělíme na 2 části do dvou podmínek.

```
if(other.type === "enemy"){
// Zde uvedeme celý předešlý kód
```

```
}
else if(other.type === "thorn"){
// Zde nastavíme nový kód pro jiný nepřátelský objekt
}
```

Kód pro objekt „thorn“, čili ThornEntity, máme již vytvořený pouze z konstruktoru a primární kolizi s hráčem jsme vypnuli. Což byl účel, abychom mohli dosáhnout efektu, že při prvním styku hráče s tímto objektem, vynutíme skok, ubereme pokusy či životy a rozblikáme hráče. Pokud by v době problikávání hráč znovu skočil na tento objekt, nic se nestane a to proto, že jsme tu primární kolizi s hráčem vypnuli v objektu ThornEntity.

Takto vypadá tedy kód, který bude v těle podmínky `other.type===“thorn“`:

```
if(!this.renderable.isFlickering() && game.data.life>1){
            this.body.falling = true;
            this.body.vel.y   =   -this.body.maxVel.y   *
me.timer.tick;
            this.body.jumping = true;
            game.data.life-=1;
            this.renderable.flicker(5000);
}
if(!this.renderable.isFlickering() && game.data.life==1){
          game.data.life=0;
          me.game.world.removeChild(this);
}
```

# 12 Základní informace HUD

Nyní se zaměříme na tvorbu zobrazování informací na obrazovce při hře. Což znamená zobrazení dvou základních věcí a to stav pokusů (život) a stav kreditů (skóre). Pro tento účel vytvoříme nový soubor HUD.js.

Abychom mohli zobrazovat nějaké informace přímo na displeji, tak potřebujeme vytvořit objekt, který je v MelonJS pojmenován jako „kontejner“.

```
game.HUD.Container = me.Container.extend({});
```

Toto pojmenování je logické, protože je to jakýsi balíček, objekt či vrstva, která může obsahovat více dalších objektů, jejichž konstruktory jsou na jiném místě a v tomto HUD objektu jsou volány a vytvářeny.

Konstruktor objektu má pouze inicializační funkci, ve které je několik málo informací o tomto objektu, jako je jméno, pozice Z, zda je plovoucí a zda nezmizí z obrazovky při změně levelu.

```
game.HUD.Container = me.Container.extend({
        init: function() {
                this._super(me.Container, 'init');
                this.isPersistent = true;
                this.floating = true;
                this.z = Infinity;
                this.name = "HUD";
        }
});
```

Uvnitř této inicializační funkce budeme později volat další konstruktory, které se zapíší následujícím způsobem:

```
this.addChild(new game.HUD.ScoreItem(750, 10));
this.addChild(new game.HUD.LifeItem(230, 10));
```

Toto jsou dva konstruktory pro zobrazení bodů a pokusů/života. V tomtéž souboru HUD.js přejdeme k vytváření těchto objektů.

```
game.HUD.ScoreItem = me.Renderable.extend( {
     init: function(x, y) {
     this.isPersistent=true;
     this._super(me.Renderable, 'init', [x, y, 10, 10]);
     this.font = new me.BitmapFont("32x32_font", 32);
     this.font.set("right");
     },
     draw : function (renderer) {
          this.font.draw      (renderer,'KREDITY:'      ,
this.pos.x, this.pos.y);
          this.font.draw    (renderer,    game.data.score,
this.pos.x+100, this.pos.y);
     }
});
```

Oba objekty ScoreItem a LifeItem vytvoříme stejným způsobem, protože se jedná o objekty, ve kterých budeme vykreslovat pomocí bitmapového fontu 32x32font.png, proto vedle inicializační funkce obsahují i funkci `draw()`. Tyto dva objekty vykreslují hodnotu na displej podle zvolené globální proměnné a to tedy podle `game.data.score` a `game.data.life`.

Jejich pozice na obrazovce měníme přes konstruktor volaný v inicializační funkci kontejneru HUD, při jejich volání jim předáváme číselné parametry pro osu X a Y, které znamenají pozici na herní obrazovce v pixelech.

Protože jsme vytvářeli nový soubor musíme jej přidat do index.html:

```
<script type="text/javascript" src="js/entities/HUD.js">
</script>
```

# 13 Registrování objektů při načítání hry

Pokud vytváříme nový objekt zejména v entities.js a jedná so o objekt definovaný i v Tiled editoru, musíme jej přidat při načítání hry do herního světa. v našem případě se jedná o nepřátelské objekty EnemyEntity, FlyingEntity a ThornEntity. Předměty ke sbírání HealthEntity, Kredit2Entity a další.

V souboru game.js bude pak zápis ve funkci `loaded()` vypadat takto:

```
me.pool.register("mainPlayer", game.PlayerEntity);

me.pool.register("Kredit2Entity", game.Kredit2Entity);

me.pool.register("Kredit4Entity", game.Kredit4Entity);

me.pool.register("Kredit6Entity", game.Kredit6Entity);

me.pool.register("EnemyEntity", game.EnemyEntity);

me.pool.register("HealthEntity", game.HealthEntity);

me.pool.register("FlyingEntity", game.FlyingEntity);

me.pool.register("ThornEntity", game.ThornEntity);
```

# 14 Vytvoření nové obrazovky

## 14.1 Stavy herní obrazovky

Stavy obrazovky či stavy hry slouží k tomu, abychom mohli zobrazit i jiné informace na herní obrazovce po načtení hry. Zejména slouží k tomu, abychom si hru rozdělili do několika fází, které se v herním průmyslu běžně používají. Jedná se o stav nějakého menu, zobrazení informací o ovládání hry, zobrazení informací po dohrání hry a další.

V základu je k dispozici 10 následujících stavů hry:

- CREDITS
- GAME_END
- GAMEOVER
- LOADING
- MENU
- PLAY
- READY
- SCORE
- SETTINGS
- USER

Poslední stav USER je určen pro definování dalších uživatelských stavů vkládáním tohoto stavu do proměnné a přičítáním libovolných hodnot:

```
var STATE_INFO = me.state.USER + 0;
```

V našem případě to ale nebude potřeba takto využít k naší hře postačí pár herních stavů. Abychom mohli volat tyto stavy, přepínat mezi stavy a další věci, je třeba je určit při načítání hry, podobně jako registrování objektu do hry a to ve funkci `loaded()` v game.js.

Zde nastavíme stavy tak, že zavoláme funkci me.state.set(), která očekává atributy jméno stavu a konstruktor pro danou obrazovku.

Toto jsou všechny stavy, obrazovky, které bude mít naše hra:

```
me.state.set(me.state.MENU, new game.TitleScreen());
me.state.set(me.state.PLAY, new game.PlayScreen());
me.state.set(me.state.SETTINGS,new game.InfoScreen());
me.state.set(me.state.END, new game.OutroScreen());
me.state.set(me.state.CREDITS,new game.StoryScreen());
```

### 14.1.1 Funkce onResetEvent()

Většina dosavadních objektů se vytvářela funkcí inicializační. Zde se jedná o jiný typ objektu „ScreenObject“. Herní manažer při změně obrazovek, při načtení hry, resetování hry volá funkci `onResetEvent()` dané obrazovky. Nyní začneme vytvářet novou úvodní obrazovku „TitleScreen“.

```
onResetEvent : function() {
   me.state.transition("fade", "black", 500);
   me.game.world.addChild(       new       me.Sprite(0,0,
me.loader.getImage('titleScreen')),1 );
},
```

Zde si můžeme nastavit plynulé prolínání obrazovek pomocí funkce `transition()`. Obdobně jako u HUD zde dochází k případnému přidání dalších objektů či volání konstruktorů. Pomocí funkcí `addChild()`, `meSprite()` a `getImage()` si přidáme obrázek titleScreen.png, který musí být též uveden v resources.js a ten se vykreslí při změně na tento stav hry.

### 14.1.2 Funkce onDestroyEvent()

Tato funkce se volá, pokud z daného stavu, obrazovky, odcházíme. Zde se má zajistit zničení vytvořených objektů, zastavení hudby, odmapování tlačítek a jiných věcí, které se vytvářely při načítání dané obrazovky.

```
onDestroyEvent : function() {
     me.input.unbindKey(me.input.KEY.ENTER);
```

```
        me.input.unbindPointer(me.input.mouse.LEFT);
        me.event.unsubscribe(this.handler);
        me.game.world.removeChild(this.MusicUI);
        me.game.world.removeChild(this.InfoPage);
        me.game.world.removeChild(this.StoryPage);
        me.audio.stopTrack();
    }
```

Toto je příklad kódu funkce, ve kterém dochází k odmapování tlačítka myši a klávesy enter, odebrání objektů z obrazovky a poslední je zastavení hrající hudby.

### 14.1.3 Vykreslení textu na obrazovce

K vykreslení textu na obrazovce může sloužit buďto bitmapový font a nebo systémový font, jak jsem již zmiňoval v kapitole se zobrazováním základních informací a HUD. To jsme vytvářeli konstruktor pro vykreslení bitmapového textu a pak jej zavolali v kontejneru HUD. Nyní si zkusíme vykreslit text systémový a to přímo ve funkci `onResetEvent()`.

Budeme přidávat nový vykreslovací objekt v této funkci a rovnou jej i rozvedeme a nastavíme v této funkci. To znamená, že opět dojde na funkce `init()` a `draw()`.

```
me.game.world.addChild(new (me.Renderable.extend ({
    init : function() {
        this._super(me.Renderable, 'init', [0, 0, me.game.viewport.width, me.game.viewport.height]);
        this.font = new me.Font('Helvetica Neue', 40, "black");
    },
    draw : function (renderer) {
this.font.draw(renderer.getContext(), "Stiskni 'ENTER' a hraj!", 240, 240);
    }
```

```
})), 2);
```

Zde nastavíme font na Helvetica Neue černé barvy a velikosti 40. Ve vykreslovací funkci jej použijeme a vykreslíme text „Stiskni ENTER a hraj!“.

### 14.1.4 Změna obrazovek

Abychom mohli z dané načtené obrazovky odejít, čili ji přepnout na další herní obrazovku musíme ve funkci `onResetEvent()` nastavit tlačítka na klávesnici a zavolat funkci posluchače `event.subscribe()`, aby hlídal, zda nedošlo ke stisku tlačítka pro odchod z této obrazovky.

```
me.input.bindKey(me.input.KEY.ENTER, "enter", true);

me.input.bindKey(me.input.KEY.I, "I", true);

this.handler    =    me.event.subscribe(me.event.KEYDOWN,
function (action, keyCode, edge) {
            if (action === "enter") {
                me.state.change(me.state.PLAY);
            }if (action === "I") {
                me.state.change(me.state.SETTINGS);
            }
});
```

Zde jsme nastavili a pojmenovali tlačítka enter a tlačítko I. a posluchač hlídá, které z těchto tlačítek bylo stisknuto a podle toho změní obrazovku. Pokud byl stisknut enter, změníme obrazovku na herní obrazovku, neboli PLAY obrazovku. Pokud bylo stisknuto tlačítko I, změníme na nějakou informační obrazovku s ovládáním, tedy obrazovku SETTINGS.

Toto je tedy změna čili odchod z dané obrazovky na jinou. Měnit obrazovky lze ale kdykoli v průběhu hry. Stačí na příslušném řádku kódu zavolat změnění obrazovky a dojde ke změně. K tomu stačí pouhá řádka kódu s příslušným jménem obrazovky, kterou žádáme:

```
me.state.change(me.state.SETTINGS);
```

Je nutné zmínit, že první a ta nejdůležitější změna obrazovky je ihned při načítání hry a to ve funkci `loaded()` v souboru game.js. Na konci této funkce přecházíme ze stavu načítání (loading) na obrazovku či stav, již pro hru (play). Pokud nyní zkontrolujeme konec této funkce, je zde nastavená změna na herní obrazovku, tedy `me.state.change(me.state.PLAY)`. Což je vlastně ten důvod, že po načtení stránky index.html v prohlížeči, ihned začne hra s načtenou mapou z Tiled editoru apod.

Pokud již máme titulní obrazovku vytvořenou, či nějakou jinou obrazovku, kterou bychom chtěli zobrazit ještě před spuštěním levelu, musíme provést změnu obrazovky již právě ve funkci `loaded()` v game.js.

Pokud tedy máme titulní obrazovku přidanou v též funkci: `me.state.set(me.state.MENU, new game.TitleScreen());`

Pak na tuto obrazovku můžeme přejít na konci této funkce. Místo stavu PLAY bude tedy stav MENU:

```
me.state.change(me.state.MENU);
```

# 15 Hudba a zvukové efekty

Na začátku tvorby hry jsme měli v souboru game.js následující řádek kódu:

```
me.audio.init("mp3,ogg");
```

Což zajistilo inicializaci, připravenost, zvukových zdrojů, o které se stará implementovaný zvukový ovladač, mixér, Howler.

V závorce jsou uvedeny formáty, ve kterých bychom měli mít naše zvukové efekty a písničky. Každý webový prohlížeč si vybere zvukový formát, který podporuje, a který mu vyhovuje. Doporučuji mít všechny písničky a zvukové efekty duplikátně v obou formátech. Mohlo by se stát, že hra bude bez zvuku, pokud jí spustíte v prohlížeči jen s podporou OGG a hudební zdroje budou ve formátu MP3.

Vlastní přidání hudby je velice jednoduché. v souboru resources.js pouze připíšete požadovaný soubor s lokací.

```
{name: "intro_music", type: "audio", src: "data/bgm/"},
```

A poté již lze spustit hudbu či zvukový efekt opět z jakéhokoli řádku kódu. Spustit hudbu lze dvěma způsoby. Slouží k tomu tyto dvě funkce `play()` a `playTrack()`.

```
me.audio.playTrack("intro_music");
```

```
me.audio.play("jump");
```

Obě funkce udělají totéž, ale přesto se využívají jiným způsobem. Pokud chceme zahrát nějaký krátký zvukový efekt, který by měl být velice krátký (zhruba do 3 sekund), pak použijeme funkci `play()`.

Pokud chceme nechat zahrát hudbu, která bude podbarvovat prostředí hry, použijeme funkci `playTrack()`. Oproti předchozí funkci toto přehrávání můžeme pozastavit což u předešlé funkce nelze: `me.audio.pauseTrack("intro_music");`

Zde je příklad spuštění zvukového efektu po sebrání kreditu:

```
onCollision : function (response, other) {

     switch(other.body.collisionType){

       case me.collision.types.PLAYER_OBJECT:
```

```
        me.audio.play("cling");
       game.data.score += 6;

this.body.setCollisionMask(me.collision.types.NO_OBJECT);
       me.game.world.removeChild(this);
       return false;
       break;
}
```

Můžeme si všimnout, že spouštíme zvukový efekt cling.mp3/ogg ve funkci `onCollision()`. v této funkci právě dochází k sebrání objektu a v ten moment chceme, aby se zahrál zvukový efekt, tudíž tam vložíme `me.audio.play(„cling“)`. Stejným způsobem můžeme přidat zvuk pro skok jump.mp3/ogg, zvuk při skoku na nepřátelský objekt stomp.mp3/ogg a další.

# 16 Herní efekty

V rámci frameworku je i implementovaný tzv.: Particle Emittor (Particles) [23], který slouží k vytvoření herních efektů. V mé hře je tento plugin použit k simulaci deště.

Autorem tohoto pluginu je herní vývojář Andre Antonio Schmitz, který působí ve známé herní společnosti Cian Games[7].

K vytvoření a libovolné editaci herních efektů můžeme využít tuto stránku: http://melonjs.github.io/examples/particles/, která patří pod MelonJS GitHub repozitář. Na dané stránce nalezneme přehledný editor herních efektů, které lze upravovat pomocí posuvníků a v reálném čase vidět výsledný efekt. Na stránce je zároveň generátor výsledného kódu efektu, který lze zkopírovat a vložit na místo v naší hře, kde chceme, aby se efekt provedl.

Jak jsem již zmínil, ve své hře jsem použil efekt, který jsem upravil a docílil tak simulace deště. Kód jsem ale záměrně zakomentoval, aby nedocházelo ke zpomalení plynulosti hry, protože se jedná o generování 200 a více malých objektů, které zpomalují vykreslování hry na pomalejších zařízeních, zejména pak mobilních zařízeních.

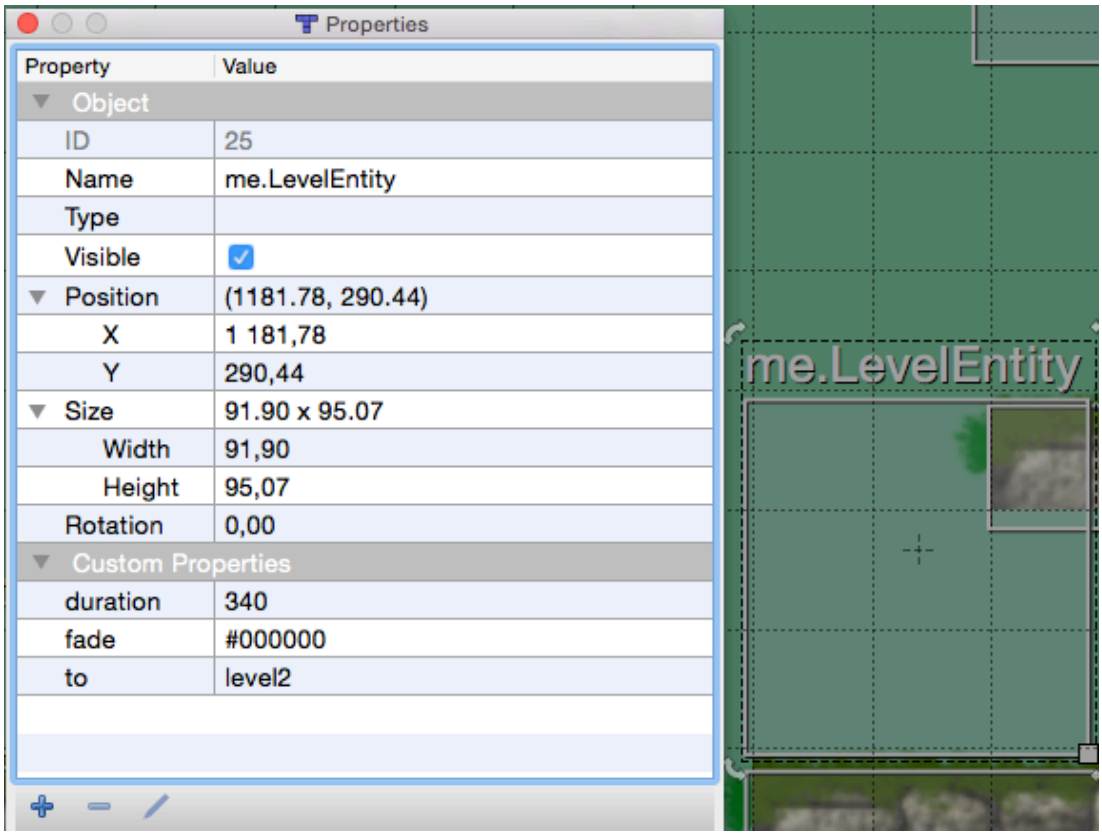
[7] Cian Games – herní společnost http://www.ciangames.com

# 17 Změna na jiný level

Abychom docílili změny levelu, mapy či lokace ve hře, je třeba mít alespoň dva levely vytvořené v Tiled editoru. Mimo všech ostatních objektů v prostředí levelu, je třeba vytvořit jeden další objekt. Tento objekt bude sloužit ke změně na další soubor s levelem.

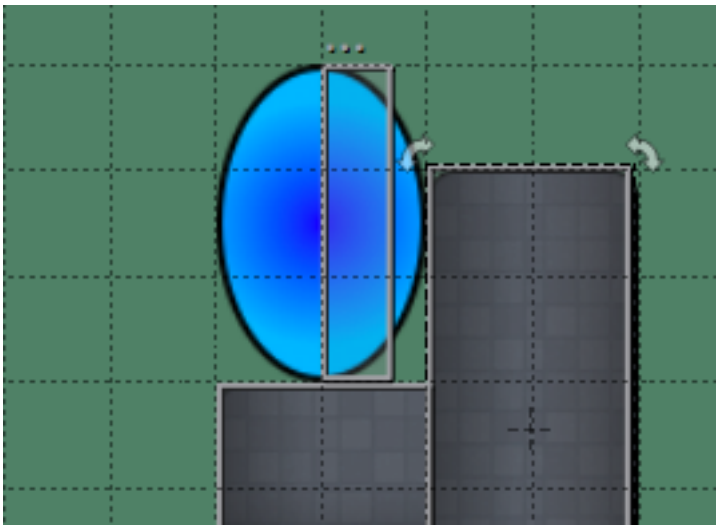
V Tiled editoru si otevřeme první level, který se spustí první při načtení hry. Vytvoříme objekt a je důležité, aby se jmenoval „me.LevelEntity“. Díky tomu framework pozná, že se bude měnit level na jiný.

Dále musíme tomuto objektu přidat atributy. Atribut „duration“ určuje, jak dlouho bude trvat přechod mezi levely v milisekundách, atribut „fade“ určuje přechodovou barvu mezi levely pomocí RGB hodnoty a poslední atribut „to“ obsahuje jméno levelu, na který má hra přejít. Teoreticky lze tento objekt mít pouze s atributem „to“, ostatní dva atributy slouží především ke zlepšení estetického dojmu.




**Obrázek 25 –** Objekt pro změnu levelu

Jedná se vlastně o objekt, který při tomto nastavení nebude na mapě vidět. Pokud se jej ale hráčova postava dotkne, dojde ke změně levelu. Více nastavení nemusíme řešit, snad jen, že pozadí objektu pro změnu levelu můžeme vyplnit dlaždicemi z tilesetu, aby jej hráč dobře poznal, podobně jako na obrázku níže.



**Obrázek 26 –** Pozadí objektu pro změnu levelu

# 18 Spuštění na více platformách

Podle dosavadního postupu jsme vytvářeli hru hlavně pro lokální PC. Hra je spustitelná ve webovém prohlížeči a přenosná maximálně zkopírováním dat kupříkladu na USB flash disk.

Doporučuji hru nahrát na internet a využít služby poskytovatelů hostingu a domén. Kupříkladu Active24, Wedos, DataHousing a další. Hru samozřejmě můžeme nahrát i na neplacený hosting, který poskytuje Internetcentrum, Webzdarma, Endora a další. Tuto cestu neplacených hostingů bych nedoporučoval, protože se všude bude zobrazovat reklama, která nám později může a nejspíše i bude vadit při spouštění na mobilních zařízeních.

Další věcí, která se může projevit při spouštění na starších PC, monitorech, či při streamování obrazu na televizi přes AppleTV a jiné smart zařízení je, že efekty ve hře nebudou tak rychlé, nemusí být tak dynamické jako na PC. Testoval jsem například funkci, kdy postava hráče problikává přes funkci flicker() a hráč buďto zmizel z obrazovky, nebo stál nedotčený. Díky defaultnímu nastavení FPS na hodnotu 60 vlastně při pomalejšímu vykreslování či naopak u novějších TV rychlejšímu vykreslování došlo k tomu, že hráčova postava sice problikávala, ale TV či některé monitory toto zobrazí buď, že hráč není vidět anebo pávě tak, že je vidět bez problikávání.

Toto lze vyřešit tak, že manuálně snížíme herní FPS na 30 již na začátku v onload() funkci v game.js souboru.

```
me.sys.fps = 30;
```

Rychlost rámu za sekundu (FPS) lze sice libovolně nastavit prakticky na jakoukoli hodnotu, ale doporučuji tuto hodnotu nastavovat tak, aby dělila hodnotu 60 bezezbytku, tedy na hodnotu 30 anebo méně 15.

Jádro frameworku využívá funkci requestAnimationFrame() [21] k synchronizaci obrazu. Tato funkce může při jiném nastavení FPS využívat tzv. frame skipping, vynechávání rámců, která funguje samozřejmě ideálně, pokud je hodnota dělitelem 60, tedy 30 nebo 15. u jiných hodnot toto nemusí fungovat správně a mohou se

porušit i některé časované herní funkce pokud je budeme využívat ve hře, tzn. jejich čas nebude odpovídat skutečnosti apod.

## 18.1 Spuštění na mobilních zařízeních

Pokud máme hru nahranou na nějakém webovém hostingu a lze k ní přistoupit přes klasický webový prohlížeč pak ji lze spustit i v mobilu. Dnešní smart zařízení, ať už mobily či tablety, které běží na majoritních operačních systémech (iOS, Android, Windows Phone), mají ve svých prohlížečích podporu HTML5. Hra tedy půjde pomocí prohlížeče spustit, ale již nepůjde ovládat, protože je nastavená na použití klávesnice.

Abychom mohli hru ovládat na dotykovém zařízení, musíme se samozřejmě přizpůsobit dotykovému displeji, přidat tlačítka na herní obrazovku, přizpůsobit velikost displeje apod.

Nejjednodušší možnost jak zajistit velikost herní plochy je ponechat dosavadní nastavení velikosti herního okna s automatickým zvětšováním podle velikosti okna prohlížeče (auto scale). Toto již máme nastavené v `onload()` funkci souboru game.js:

```
me.video.init('screen', me.video.CANVAS, 853, 480, true, 'auto')
```

Pokud nemáme již z dřívějška definováno, pak do index.html souboru nalinkujeme stylový soubor CSS, ve kterém definujeme styly pro element `<body>` a pak pro herní obrazovku `<canvas>` [7]:

```
body {
     background-color: #000;
     color: #fff;
     -moz-user-select: none;
     -ms-user-select: none;
     -o-user-select: none;
     -webkit-user-select: none;
```

```
    user-select: none;
    -ms-touch-action: none;
    touch-action: none;
    margin: 0;
}
canvas {
    margin: auto;
    display: block;
    image-rendering: optimizeSpeed;
    image-rendering: -moz-crisp-edges;
    image-rendering: -o-crisp-edges;
    image-rendering: -webkit-optimize-contrast;
    image-rendering: optimize-contrast;
    -ms-interpolation-mode: nearest-neighbor;
}
```

Nastavíme okolí herní obrazovky na černé a vypneme dotykové funkce v elementu `<body>`. Element `<canvas>` dáme do bloku, vycentrujeme, a zamezíme zásahu do herní grafiky, aby nedocházelo k vyhlazování apod. Toto je zatím nejlepší možný zápis, který platí napříč platformami a prohlížeči.

Poslední úprava v index.html souboru, která je v základu v boilerplate zmíněna, je tento skript:

```
<script type="text/javascript">
    window.onReady(function onReady() {
    game.onload();
    if (me.device.isMobile && !navigator.isCocoonJS) {
    window.document.addEventListener("touchmove",
function (e) {
```

```
            e.preventDefault();
            window.scroll(0, 0);
            return false;
        false);
         (function () {
                window.scrollTo(0, 1);
                me.video.onresize(null);
            }).defer();
        me.event.subscribe(me.event.WINDOW_ONRESIZE,
function (e) {
                window.scrollTo(0, 1);
        });
        }
});
</script>
```

Zde ještě pomocí javascriptu zamezíme rolování stránky pro mobilní zařízení a vynutíme schování navigačních a jiných lišt v prohlížeči, tak aby se hra mohla roztáhnout na celý displej.

Toto jsem testoval na zařízeních jako je iPhone 5, iPhone 6, iPad mini retina, Samsung Galaxy S5, Xiaomi Mi3. Bohužel mezi vývojáři jednotlivých operačních systémů a prohlížečů neplatí, žádný standard a tak toto funguje různě.

Nejlépe dopadly zařízení od Applu s iOS, kdy se prohlížeč poctivě schová po otočení mobilu do vodorovné polohy (landscape). Pokud se ale dotkneme níže ke dolnímu kraji displeje, vyjedou ovládací lišty opět napovrch a musíme otočit zařízení znovu do svislé polohy a ihned do vodorovné polohy, aby se zas schovaly. Logicky funkce prohlížeče má větší prioritu, před tímto skriptem. Na druhou stranu, ostatní testované zařízení s Android OS ovládací lišty neschovaly vůbec.

## 18.2 Ovládání pro mobilní zařízení

Abychom mohli přidat ovládání pro mobilní zařízení, je nutné nejdříve vědět, zda je hra právě zobrazována na herním zařízení. Pro tento účel má framework jednu velice užitečnou funkci `me.device.isMobile()`, která vrací `true` nebo `false`, podle toho zda je hra (stránka) zobrazována na mobilním zařízení či není. Tato funkce je klíčem ke všem dalším krokům, protože ve hře naprogramujeme tlačítka a jiné zobrazení obrázků, ale zobrazíme je až v době, kdy nám tato funkce vrátí `true`, že se jedná o mobilní zařízení.

Nejdříve si připravíme obrázky tlačítek z adresáře „onscreenControls“. Tato tlačítka jsou též dostupná na adrese: http://opengameart.org/content/onscreen-controls-8-styles. v tomto balíčku najdeme soustu obrázků tlačítek vytvořených pro účely her pro mobilní zařízení. Vybereme proto obrázky šipek doprava, doleva a obrázek tlačítka pro skok, a přidáme je do resources.js.

Pokud hru nyní načteme v mobilním prohlížeči a máme nastavenou titulní stranu (`me.state.MENU`) pak se samozřejmě vše v pořádku zobrazí. Na PC bychom přešli do herní obrazovky stiskem klávesy enter. Na mobilním zařízení se ale nic nestane. Proto již zde na začátku v konstruktoru pro titulní stranu budeme přidávat funkci, která rozpozná dotek prstu na displej pomocí funkce `bindPointer()`:

```
me.input.bindPointer(me.input.mouse.LEFT,
me.input.KEY.ENTER);
```

Ve volném překladu funkce mapuje ukazatele, čili souřadnice X a Y na displeji, kam jsme položili prst. Do této funkce pak dáme dva atributy pro simulaci stisku levého tlačítka myši a tlačítka enter. To znamená, že pokud se dotkneme displeje framework nasimuluje událost, jako kdyby došlo ke stisku těchto tlačítek. Tento řádek přidáme do funkce `onResetEvent()` na místo, kde mapujeme ostatní klasická hardwarová tlačítka. Když se nyní po načtení hry dotkneme displeje, hra přepne obrazovku na herní.

Nyní když se ocitneme v herním stavu obrazovky (me.state.PLAY), budeme zde volat konstruktory pro tlačítka opět ve funkci `onResetEvent()`. Nesmíme zapomenout na podstatu věci a tento konstruktor obalit podmínkou pokud se jedná

o mobilní zařízení. Zároveň jej poté v `onDestroyEvent()` funkci musíme odebrat. Kód vypadá následovně:

```
if(me.device.isMobile){
     this.MobileHUD = new game.MobileHUD.Container();
     me.game.world.addChild(this.MobileHUD);
}
```

Konstruktor objektu MobileHUD sice již voláme, ale není vytvořený. Vytvoříme si tedy nový soubor, který pojmenujeme MobileUI.js či MobileHUD.js a nalinkujeme jej do index.html.

Vlastní herní HUD jsme již vytvářeli a zde se bude postupovat podobným způsobem. Opět se jedná o kontejner, který zajistí vrstvu na herní obrazovce pro vytvoření třech tlačítek:

```
game.MobileHUD.Container = me.Container.extend({
init: function() {
     this._super(me.Container, 'init');
     this.isPersistent = true;
     //this.floating = true;
     this.z = Infinity;
     this.name = "MobileHUD";
this.addChild(new game.MobileHUD.rightButton(180,350));
this.addChild(new game.MobileHUD.leftButton(30,340));
this.addChild(new game.MobileHUD.jumpButton(645,335));
}
});
```

Zde si můžete všimnout, že proměnná `floating` je zakomentována. To z toho důvodu, aby při posunu mapy nedošlo k odsunu obrázků tlačítek. Nastavená citlivá plocha pro skákání a pohyb by sice zůstala na stejném místě původního obrázku

tlačítka, ale samotný obrázek by odeplul z herní obrazovky. Ostatní nastavení je stejné jako u HUD.js.

Dále zde voláme konstruktory pro tři tlačítka rightButton (doprava), leftButton (doleva) a jumpButton (skok). v závorkách pak jsou souřadnice X a Y pro pevnou pozici na displeji.

Kód pro jednotlivá tlačítka je téměř stejný. Jedná se o konstruktor, ve kterém voláme obrázek pro tlačítko dle jména obrázku z resources.js. Poté definujeme přesnou velikost obrázku v pixelech (výška a šířka). Liší se samozřejmě až simulace typu tlačítka:

```
game.MobileHUD.leftButton = me.GUI_Object.extend({

    init: function (x, y) {

       this._super(me.GUI_Object, "init", [ x, y, {

           image: "left",

           spriteheight: 130,

           spritewidth: 130

       }]);

me.input.registerPointerEvent('pointerdown',this,
function() {

     me.input.triggerKeyEvent(me.input.KEY.LEFT, true);

       });

me.input.registerPointerEvent('pointerup',this,
function() {

     me.input.triggerKeyEvent(me.input.KEY.LEFT, false);

       });

       this.z = Infinity;

},

onDestroyEvent: function() {

    me.input.triggerKeyEvent(me.input.KEY.LEFT, false);
```

```
    }
});
```

Zde ve funkci `init()`, registrujeme posluchače pro dotek na displeji pomocí `registerPointerEvent()` funkce, kterou voláme dvakrát. Jednou pro stav `pointerdown` (pokud se dotýkáme displeje) a druhou pro `pointerup` (pokud se přestaneme dotýkat displeje). v těle této funkce poté voláme spuštění události stisku tlačítka `triggerKeyEvent()`. Tato funkce nasimuluje stisknutí příslušného tlačítka, v případě tohoto tlačítka jde o šipku doleva, tedy `KEY.LEFT`. v stisknuté poloze musíme přidat atribut `true`, což značí, že je tlačítko stisknuté. A v druhém stavu jej přenastavit na `false`, aby postava hlavního hrdiny stále nechodila doleva, přestože jsme již tlačítko pustili.

Další důležitou věcí je funkce `onDestroyEvent()` těchto tlačítek. Při změně obrazovky, restartu hry apod. je důležité, aby se při odchodu z herní obrazovky zavolala tato funkce v objektu tlačítka a nastavila tlačítko na nestisknuté přes atribut `false`:

```
me.input.triggerKeyEvent(me.input.KEY.LEFT, false);
```

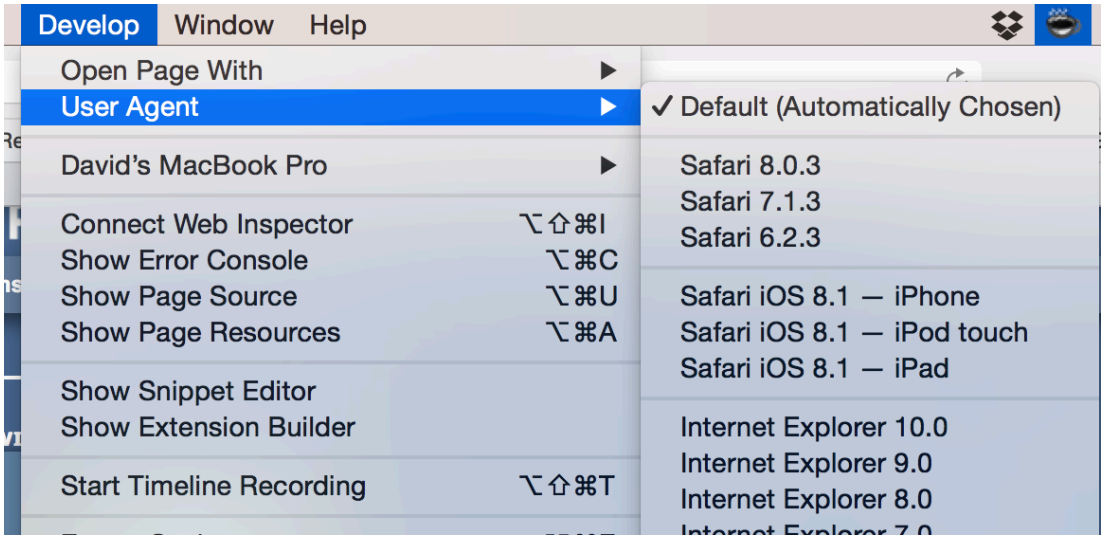
Pokud bychom tak neučinili, mohlo by se stát, že pokud by se hra ukončila v určitý moment, kdy jsme měli stisknuté tlačítko, pak by framework simuloval stisk tlačítka stále i na další obrazovce, při restartu hry apod. což se nesmí stát.

Výsledek by měl vypadat takto:


**Obrázek 27 –** Ovládací prvky na mobilním zařízení

## 18.3 Testování mobilního zobrazení

K otestování zda se tlačítka budou zobrazovat na mobilním zařízení lze využít i internetový prohlížeč. Já jsem hru testoval v prohlížeči Safari, který umí jednat jako mobilní zařízení a tak tyto situace nasimulovat aniž bychom museli hru spouštět na mobilním zařízení, pomocí nástrojů pro vývojáře a nastavení uživatelského agenta (User Agent – iPhone).




**Obrázek 28 –** Nastavení přístupu prohlížeče

## 18.4 Funkce isMobile

Funkci me.device.isMobile jsme využili pro zobrazení ovládacích prvků jen v případě, že je hra zobrazována mobilním zařízením.

Tuto funkci lze ale použít prakticky na cokoli ve hře. Lze tuto funkci volat kdekoli a tak můžeme například zobrazit i jiný obrázek na pozadí titulní strany speciálně pro mobilní zařízení, nebo můžeme vykreslit na displej „Klepni na displej a hraj!“ místo „Stiskni ENTER a hraj!“, změnit pozici pro vykreslování kreditů (skóre) a pokusů (života) a mnoho dalšího. Stačí když patřičný kód zabalíme do následující podmínky a do `if()` vložíme kód pro mobilní zařízení a do `else{}` pro PC a ostatní.

```
if(me.device.isMobile){
// kód pro mobilní zařízení
}
else{
// kód pro PC
}
```

# 19 Mobilní aplikace

Další možností spuštění vlastní hry je aplikace CocoonJS, která je též multiplatformní a ke stažení v mobilních obchodech (Apple App Store, Google play, Amazon App Store). Po stažení aplikace do mobilního zařízení je třeba se zaregistrovat pomocí libovolné emailové adresy.

Aplikace umožní načítání hry z internetu pomocí interního webového prohlížeče. Zároveň lze hru do aplikace přímo nahrát přes počítač či synchronizační program zařízení. Do aplikace zkopírujeme složku s hrou přesně tak, jako když nahráváme hru na webový hosting. Tak můžeme hru mít kdykoli po ruce a spouštět ji i bez internetového připojení.

Další výhoda aplikace je, že je schopná monitorovat tok dat, zobrazovat chyby a ukazovat FPS. Aplikace je schopná eliminovat primární vlastnosti klasického webového prohlížeče a tak se nestane, aby se zobrazila lišta s oblíbenými položkami či se zobrazilo hledání apod.

# 20 Srovnání s frameworkem Phaser

Přestože je Phaser framework vyvíjen až od roku 2013 oproti Melonu, který byl založen roku 2011, vyvíjí se rychleji. Na úvod mohu říci, že vývojářský tým o třech členech z MelonJS vyvíjí framework zároveň při své práci a snad až na malé výjimky, nemají z frameworku žádný příjem.

Kdežto Phaser má na svědomí vývojář Richard Davey a grafička Ilija Milintiević. Je to jejich hlavní práce a živí se vyvíjením her. Phaser je poskytován zdarma stejně jako MelonJS. Financování vývoje a ostatních věcí má Phaser zajištěno skrze placené konzultace, vývoj na zakázku, firemní konzultace a dohled, knihy, převod Flash her do Phaser/HTML5 podoby a mnoho dalšího včetně spolupráce se společnostmi jako je kupříkladu BBC[8] a NBC Universal[9]. [20]

| | **MelonJS** | **Phaser** |
|---|---|---|
| **Instalace** | Lokální kopie, webový server | Lokální kopie, webový server |
| **Jazyk** | Javascript | Javascript, Typescript |
| **Kompatibilita** | HTML5 prohlížeče, multiplatformní | HTML5 prohlížeče, multiplatformní |
| **Velikost** | 750KB | 2,7MB |
| **Aktuální verze** | 2.0.2 | 2.2.2 |
| **Dostupnost a aktualizace** | Github | Github |
| **Příklady kódu na webu** | 1 | 492 |
| **Příklady v balíčku** | 14 | 492 |

[8] BBC – Britská televizní a rozhlasová společnost

[9] NBC – Americká mediální společnost

| **Komplexnost a využitelnost příkladů** | Malá | Veliká |
|---|---|---|
| **Řešení herní fyziky** | Přednastavená gravitace | 3 herní styly, Box2D[10] implementace za poplatek |
| **Způsoby publikace** | Webová stránka, CocoonJS, PhoneGap[11] | Webová stránka, CocoonJS, PhoneGap |
| **Podpora** | Fórum, GitHub, online chat, email | Fórum, Github, email, twitter |
| **Knihy** | Ne | Ano (3) |
| **Online obchod** | Ne | Ano |
| **Cena** | Zdarma | Zdarma |
| **Vývojové prostředí** | Dle uživatele, Webové (Cloud9[12]) | Dle uživatele, Webové (Cloud9) |
| **Online dokumentace** | Ano | Ano |
| **Výhody** | Méně náročné programování, úzká spolupráce s programem Tiled, rychlá podpora | Více rozvinutý a profesionální framework, úspornější kód, vhodný pro vývojářský tým |
| **Nevýhody** | Pomalý vývoj | Náročnější pro začátečníky |

[10] Box2D – Fyzikální engine http://box2d.org

[11] PhoneGap – Framework, publikace hry do online mobilních obchodů http://phonegap.com

[12] Cloud9 – Webové vývojové prostředí a server http://www.c9.io

| | | |
|---|---|---|
| **Aktuality a blogy** | Ne | Ano |
| **Komunita** | Malá | Velká |

**Tabulka č. 3 –** Porovnání frameworků

Toto je porovnání současného stavu frameworků. Musím podotknout, že žádný z těchto frameworků není špatný. Hlavní faktor, který ovlivňuje tento stav je pouze na vývojářích. Je zde markantní, že vývojář Phaseru má vývoj frameworku jako svou práci a jeho každodenní náplň tvoří věci spojené s vývojem Phaseru. Toto je viditelné na četnosti příkladů a hlavně na aktivitě stránek, kde je prozatím téměř každý den publikován článek i více textů.

MelonJS se ubírá podobným směrem, pouze jeho tempo je menší z důvodu zaměstnanosti vývojářů. Pokud srovnám tehdejší stav MelonJS ve verzi 0.9 a nynější 2.0.2., pak se jedná o veliký pokrok a změnu k lepšímu jak v podpoře, tak v syntaxi kódu.

Nutno podotknout, že v době, kdy jsem začal vytvářet hru a bakalářskou práci (červenec, 2014), tak nebyla k dispozici dokumentace v takové podobě jako nyní (březen, 2015). Nebyl k dispozici aktuální příklad na stránkách a ani dalších 14 příkladů nebylo. Sice se jedná o velice slabé příklady, které nemají mnoho využití, ale je vidět, že se vývojáři postupně více zaměřují na podporu a rozvíjení příkladů.

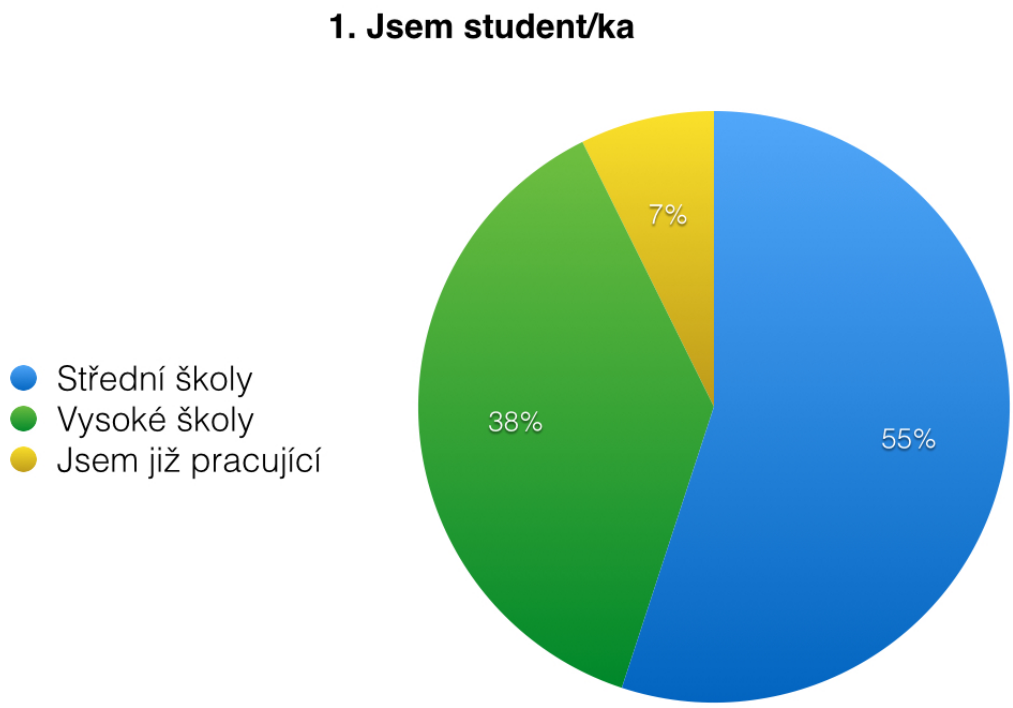
# 21 Dotazník

V rámci bakalářské práce jsem zároveň vypracoval stručný dotazník, který mi poskytl orientační informace za účelem zjištění hrubého odhadu, zda lidé znají MelonJS framework a zda by měli zájem se dozvědět více o tomto frameworku popřípadě vyzkoušet můj návod na tvorbu her.

Dotazník jsem vytvořil online pomocí služby Survio[13] a rozšířil pomocí sociálních sítí. Zároveň jsem dotazník poskytl v tištěné podobě na všeobecném osmiletém gymnáziu v Trhových Svinech studentům septimy a oktávy.

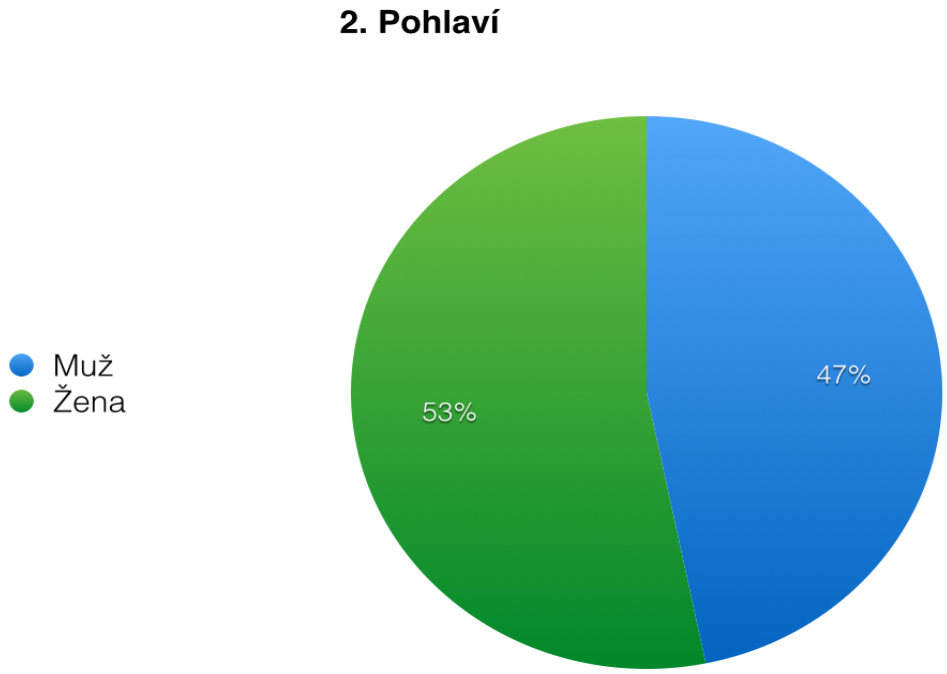
Celkem jsem shromáždil 109 respondentů.

## 21.1 Souhrn odpovědí



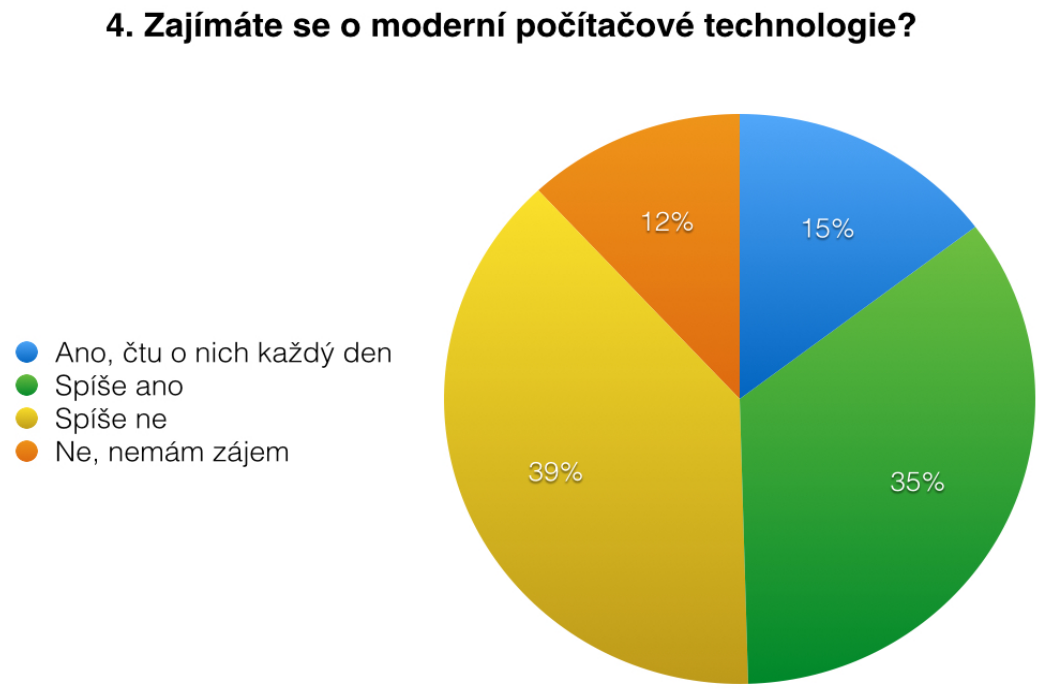

**Obrázek 29 –** Dotazník - Studium

[13] Online služba pro tvorbu dotazníkových šetření. www.survio.com

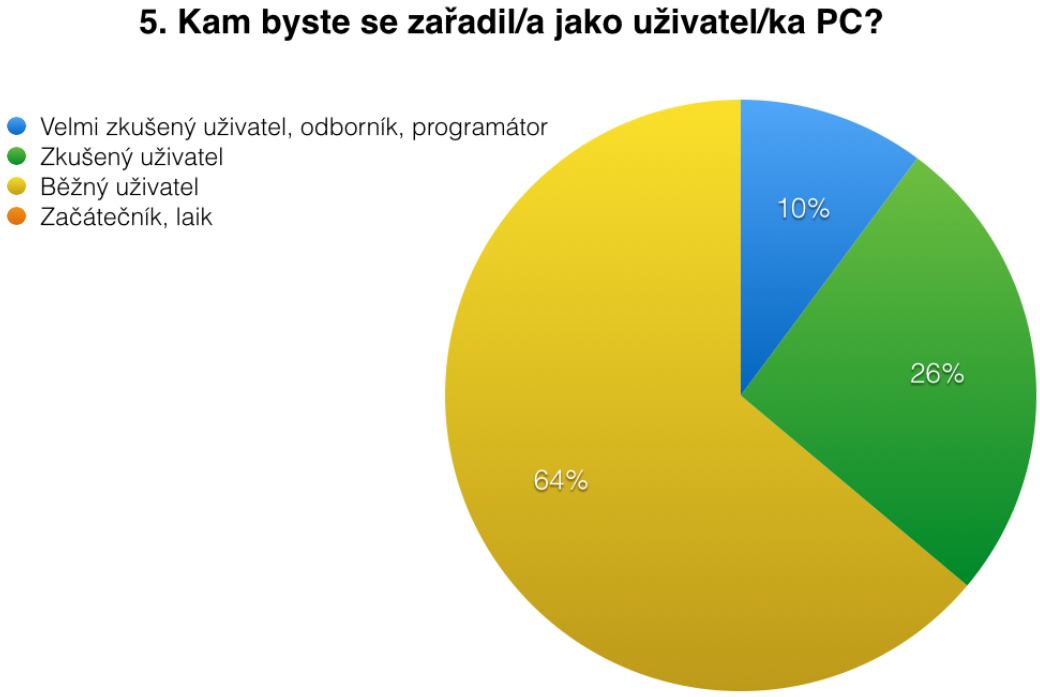



**Obrázek 30 –** Dotazník - Pohlaví

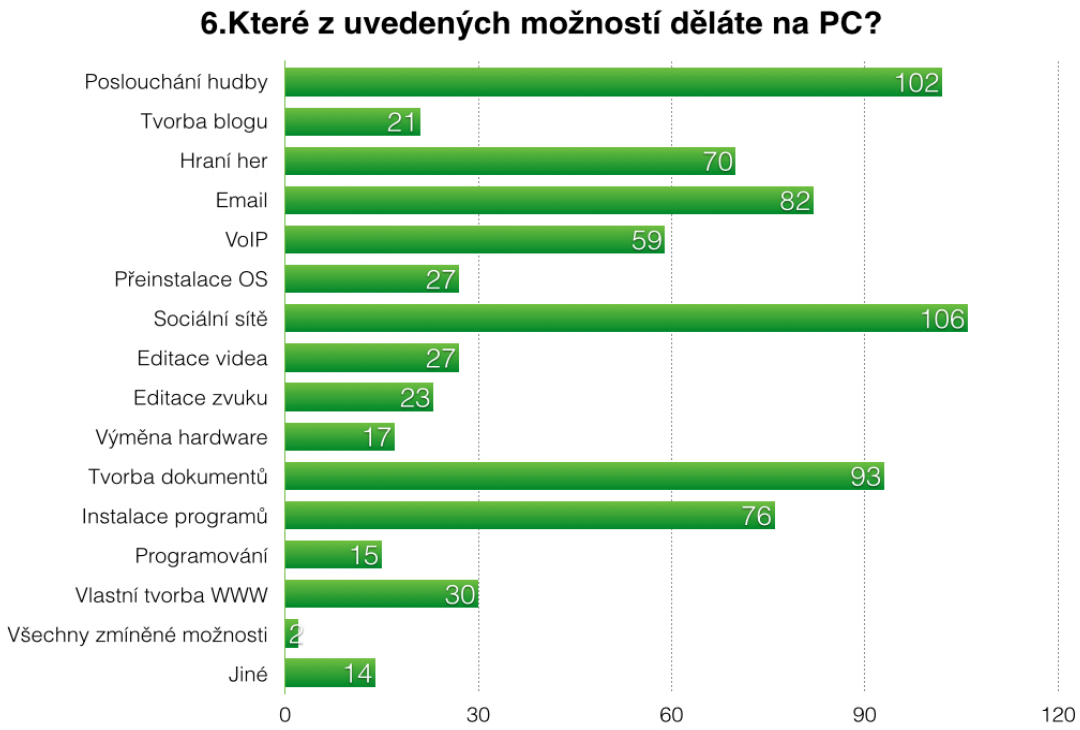

**Obrázek 31 –** Dotazník - Věk



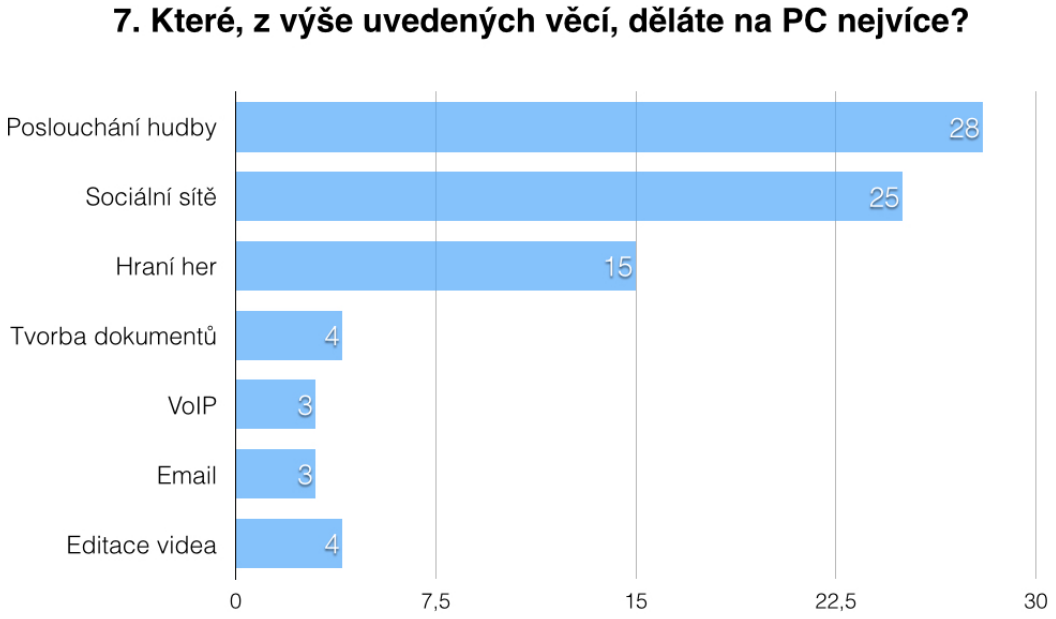

**Obrázek 32** – Dotazník - Zájem o technologie




**Obrázek 33** – Dotazník - Zkušenosti s PC

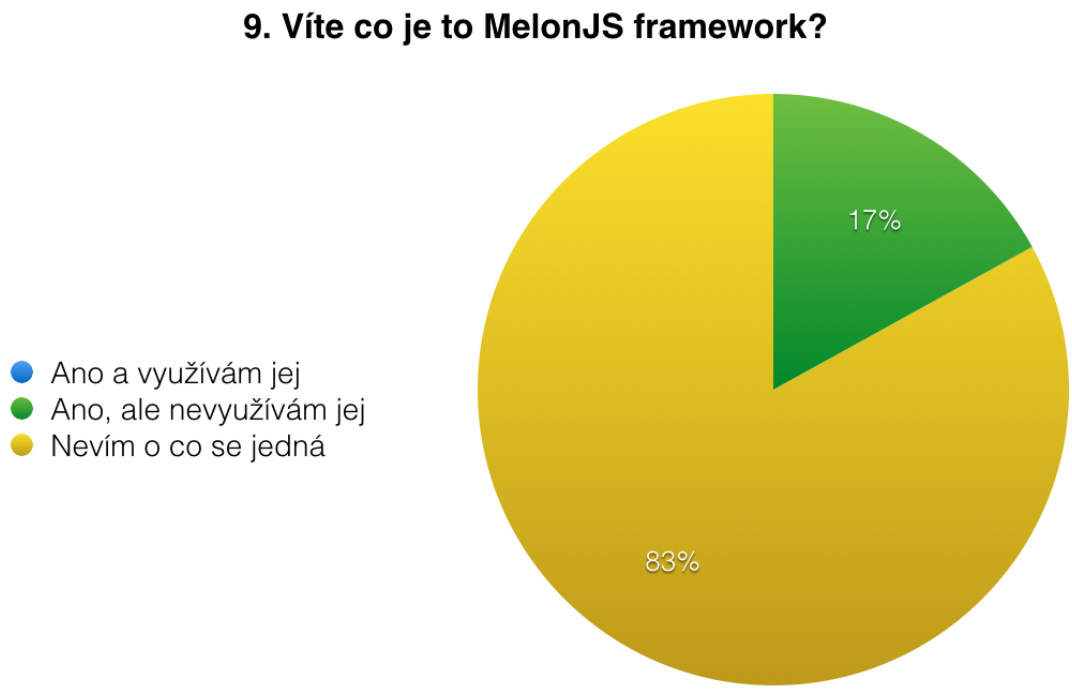



**Obrázek 34** – Dotazník - Činnosti na PC

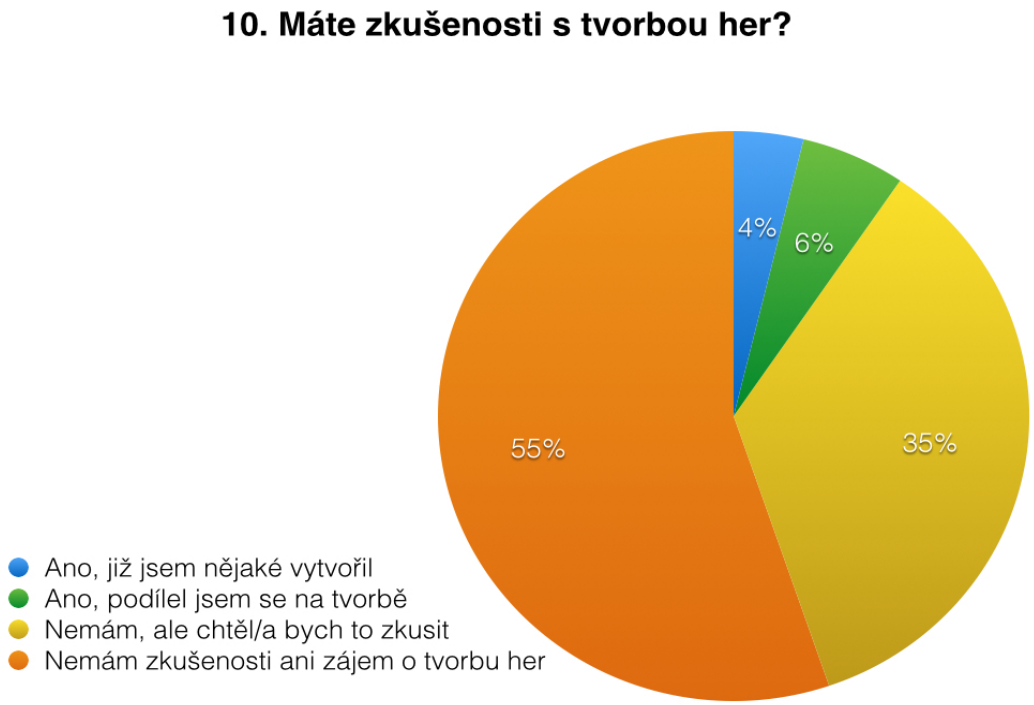



**Obrázek 35** – Dotazník - Časté činnosti na PC

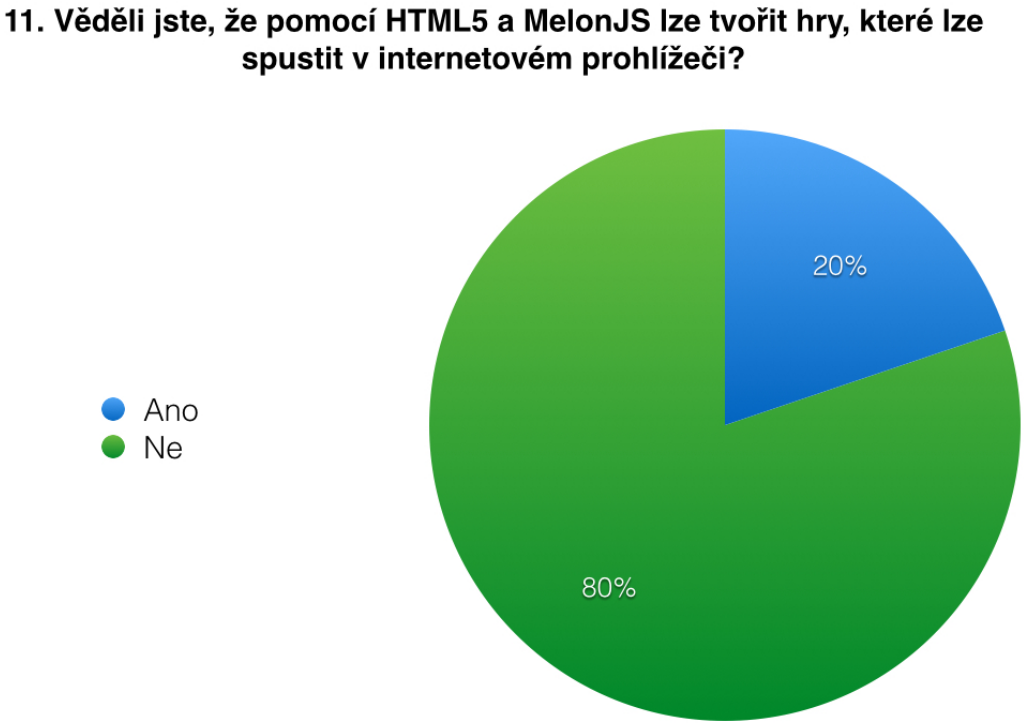

**Obrázek 36** – Dotazník - Zkušenosti s HTML5



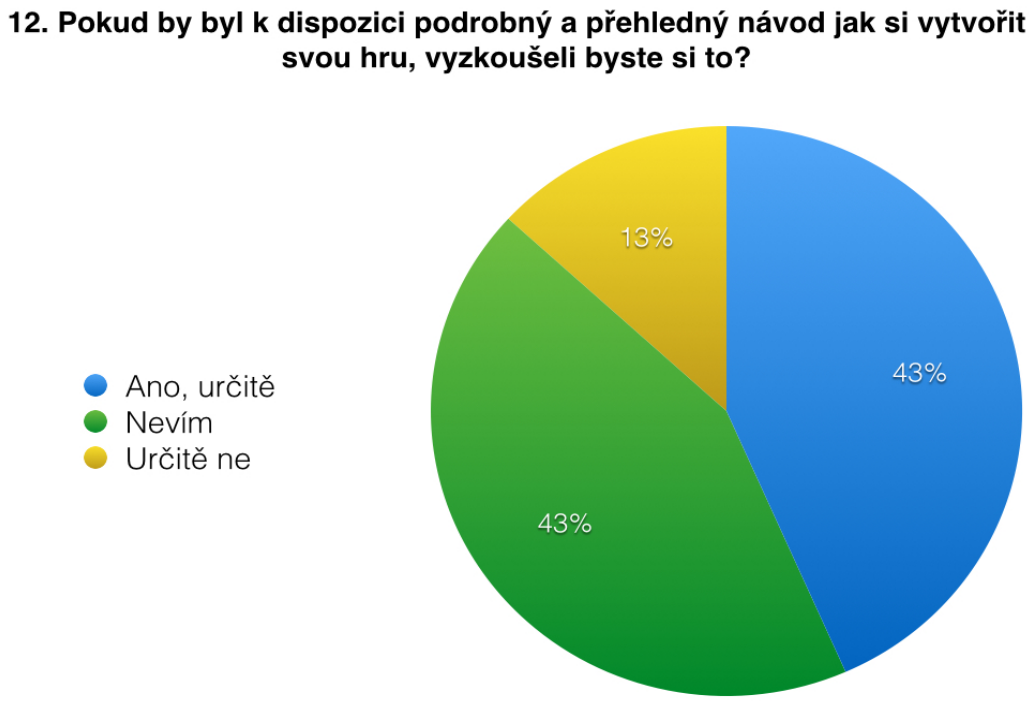

**Obrázek 37** – Dotazník - MelonJS



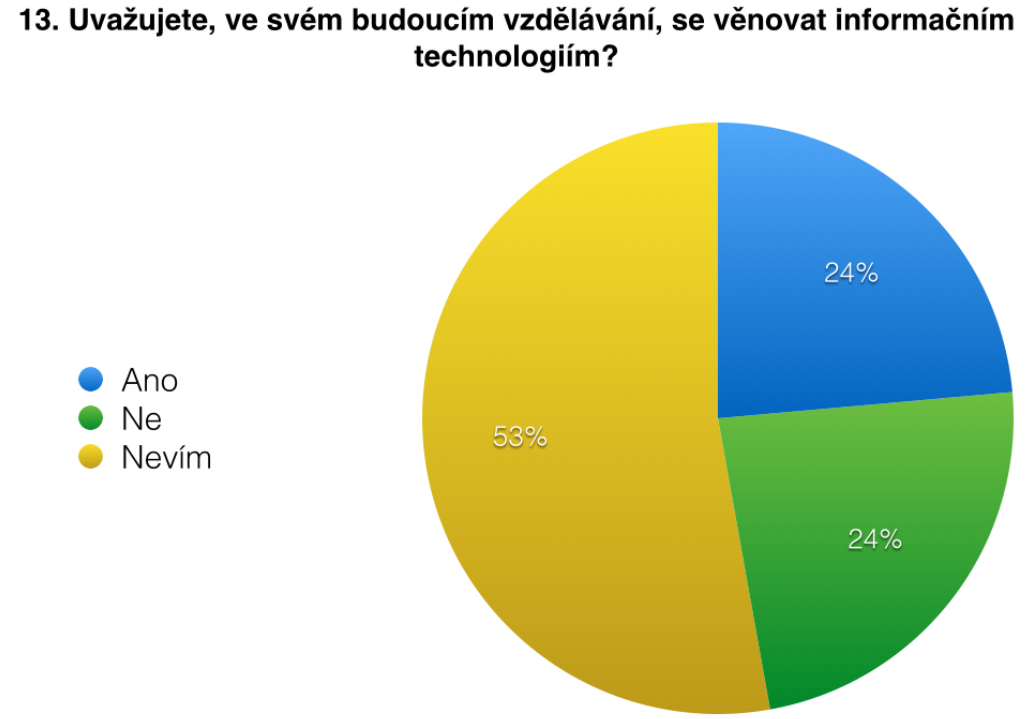

**Obrázek 38 –** Dotazník - Tvorba her




**Obrázek 39 –** Dotazník - HTML5 a MelonJS




**Obrázek 40** – Dotazník - Zájem o tutoriál




**Obrázek 41** – Dotazník - Budoucí vzdělávání

# 22 Přehled her

## 22.1 Xenon Game

V rámci mé bakalářské práce bylo cílem vytvořit multiplatformní hru. Jako svou hlavní hru, na kterou jsem kladl největší důraz je hra „Xenon Game“. Jedná se o 2D skákací hru patřící do herní kategorie „plošinovky“ či lidově „skákačky“. Úkolem hlavní postavy, hrdiny, je sesbírat 180 kreditů k dokončení hry. Hráč má zároveň k dispozici 3 pokusy. Jedná se o skrytou alegorii studenta na univerzitě, který má v rámci bakalářského studia „posbírat“ minimálně 180 kreditů, přičemž na zkoušky má 3 pokusy. v cestě mu samozřejmě stojí překážky ve formě chodících a létajících nepřátel a pozemních nástrah, které mu odčítají pokusy. Do hry jsem přidal i bonusové předměty, které hráči přidají pokus či ho učiní rychlejším.

Hru lze samozřejmě hrát na jakékoli platformě.

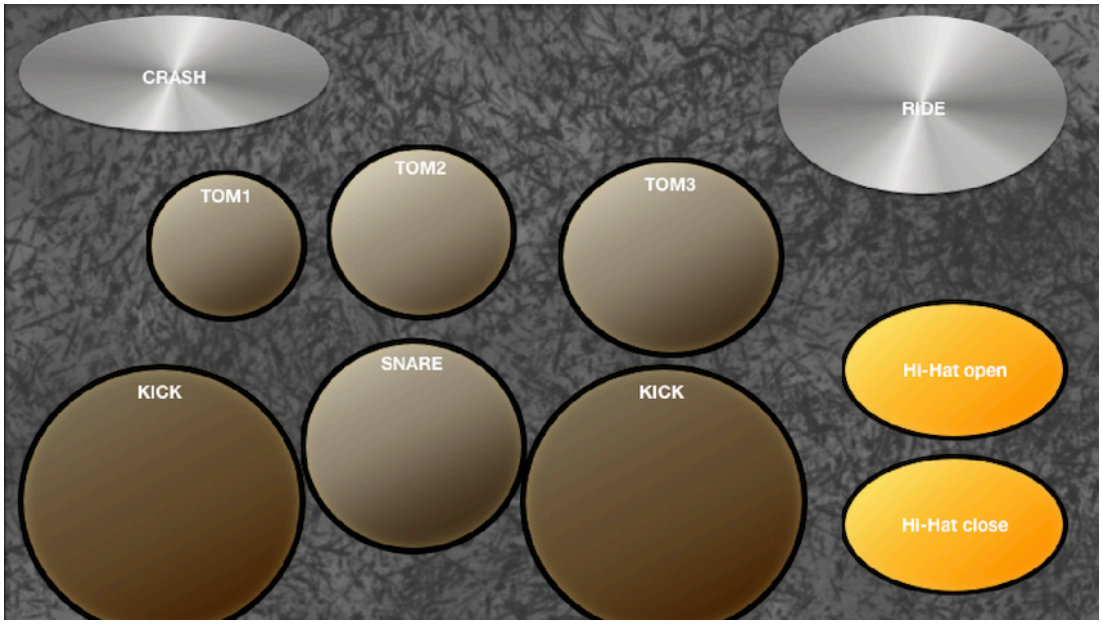

**Obrázek 42 –** Úvodní obrazovka Xenon game

## 22.2 Clumsy Drums!

S érou chytrých mobilních a hlavně dotykových zařízení přišel trend emulace hudebních nástrojů. Nastal boom ve všech virtuálních obchodech s aplikacemi typu klavír, kytara, bicí, nejrůznější syntetizéry apod.

Napadlo mě, zda lze použít tento herní framework i k podobnému účelu emulace či zvirtualizování hudebního nástroje. Jakožto bubeník jsem si vybral hudební nástroj bicí a vymyslel jsem způsob, jak lze toto udělat i s MelonJS.

Proto jsem přišel s hrou Clumsy Drums!, kterou lze spustit opět kdekoli, ale svou logikou a účelem je hlavně pro mobilní zařízení, kde uživatel používá prsty namísto paliček na bicí.




**Obrázek 43 –** Ze hry Clumsy Drums!

## 22.3 Gravity Bird

Další malou hrou, kterou jsem vytvořil navíc v rámci bakalářské práce je hra Gravity Bird. Nápadem k vytvoření této hry byla hra Flappy Bird, s kterou přišel mladý Vietnamský vývojář Dong Nguyen v roce 2013. Hru poskytl zdarma a díky reklamním bannerům ve hře vydělal obrovskou sumu peněz za velice krátkou dobu, protože hra se stala během několika dní celosvětově známou díky své obtížnosti.

Na motivy této hry jsem vytvořil hru s podobnými principy. Přičemž jsem ji přizpůsobil tak, aby byla multiplatformní a zároveň jsem si tento koncept upravil dle svého. Hra byla původně nekonečná a hráč měl původně jeden život. Mě více lákalo tuto hru uzpůsobit tak, aby zde byla možnost přidání více lokací, více obtížností skrze herní levely apod.


**Obrázek 44 –** Úvodní brazovka Gravity Bird

# 23 Závěr

Myslím si, že tvorba HTML5 s využitím dalších dostupných frameworků, v mém případě MelonJS, má dobrou budoucnost a bez pochyby vyřadí Flash technologii nejen v online hrách. Díky tomu, že lze tyto frameworky stáhnout zdarma společně s nástroji pro editaci, tak se může do vývoje her zapojit kdokoli.

Podle mého názoru lze využít framework i při výuce programování nejen javascriptu. Studenti uvidí svoji práci názorně, přičemž se jedná o nějaký reálný výstup kódu, který může být zábavný a zároveň poučný namísto strohých textových výpisů nic neříkajících údajů a hodnot. Zároveň je toto brána jak pro začínající programátorský tým specializující se na vývoj her, ale i hobby programátory, kteří mají o dané pole zájem.

Vyzkoušel a zažil jsem si, jak není lehké být vývojářem hry. Vývoj hry obsahuje spoustu rovin a aspektů, které musí jedinec řešit, přes návrh a grafiku, až po programování, testování a ladění.

Vyvíjet hru jsem, začal na verzi 0.9.x MelonJS (3/2014). Později při začínající tvorbě bakalářské práce, jsem zjistil, že během mé nečinnosti se framework dvakrát po velkých skocích zaktualizoval až na nynější verzi 2.0.2. Jednalo se o obrovské změny i v syntaxi. Proto jsem dosavadní pokrok kompletně smazal a začal od začátku na verzi 2.0.2. Začít od znovu nebylo lehké, nastaly i problémy při spuštění byť prázdného projektu protože logika a syntaxe se s novou verzí MelonJS změnila a bez jakýchkoli návodů to šlo opravdu ztěžka. MelonJS sice vypisuje chyby a jejich znění, ale vesměs se jedná o nic neříkající řetězce. v tomto začátku mi byly nápomocní vývojáři Melonu, Aron McLeod a Jay Oster, kteří mi vysvětlili co dané řetězce a chyby znamenají, až jsem projekt rozběhl a pak už to šlo bez větších problémů.

Zároveň jsem rád, že mohu být první v České Republice, kdo vytváří tutoriál pro MelonJS a zároveň jediný kdo o tomto frameworku píše. Ulehčil jsem tak dalším zájemcům práci, protože zatím neexistuje žádná publikace ani článek, který by o této problematice hovořil.

Největší nevýhodou při vytváření hry a pro PC a mobilní zařízení se jeví rozmanitost a množství zařízení. Toto se v takové míře samozřejmě netýká stolních a přenosných

PC, tak moc, jako právě mobilních zařízení. Z mých dosavadních zkušeností se s online hrou ve webovém prohlížeči nejlépe jevily zařízení firmy Apple tím, že hra šla více plynule i na starších zařízeních oproti výkonnějším zařízením od konkurentů. Přesto je hra spustitelná a hratelná všude přičemž rozeznává mobilní zařízení od nemobilního.

# Seznam použité literatury a zdrojů

[1] AUJEZDSKÝ, Josef. ROOT.CZ. Licence MIT [online]. [cit. 2015-03-09]. Dostupné z: http://www.root.cz/specialy/licence/text-licencnich-podminek-mit-s-komentarem/

[2] GITHUB. Gitter.im: MelonJS Public room [online]. [cit. 2015-03-09]. Dostupné z: https://gitter.im/melonjs/public

[3] WIKIPEDIA. *Framework* [online]. 2014. vyd. [cit. 2015-02-27]. Dostupné z: http://cs.wikipedia.org/wiki/Framework

[4] CLAY.IO. *HTML5 Game engines: Which HTML5 Game Engine is right for you?* [online]. 2015. vyd. [cit. 2015-02-27]. Dostupné z: http://html5gameengine.com/

[5] SEVENSON.COM.AU. *Separating Axis Theorem (SAT) Explanation* [online]. 2009. vyd. [cit. 2015-03-04]. Dostupné z: http://www.sevenson.com.au/actionscript/sat/

[6] SÉQUIN, Prof. Carlo H. a Jordan SMITH. BERKLEY EECS. SLIDE: Scene Language for Interactive Dynamic Environments: Geometry [online]. 2000. vyd. [cit. 2015-03-04]. Dostupné z: http://www.cs.berkeley.edu/~ug/slide/docs/slide/spec/spec_frame_geometry.shtml

[7] CSS: Image-rendering. MOZILLA DEVELOPER NETWORK. [online]. 2015 [cit. 2015-03-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/image-rendering

[8] MelonJS: a lightweight HTML5 game engine. [online]. [cit. 2014-03-16]. Dostupné z: http://www.melonjs.org

[9] Html5.cz: vše co potřebujete vědět o HTML5. [online]. [cit. 2014-03-16]. Dostupné z: http://www.html5.cz

[10] Tiled: Map editor. [online]. [cit. 2014-03-16]. Dostupné z: http://www.mapeditor.org!

[11] W3schools. [online]. [cit. 2014-03-16]. Dostupné z: http://www.w3schools.com/html/html5_intro.asp

[12] GitHub: Build software better, together. [online]. [cit. 2014-03-16]. Dostupné z: https://github.com/melonjs/melonjs

[13] THAU. Velký průvodce JavaScriptem: tvorba interaktivních webových stránek v praxi. 1. vyd. Praha: Grada, 2009. ISBN 978-80-247-2211-5

[14] PEHLIVANIAN a NGUYEN. Velký průvodce JavaScriptem: tvorba interaktivních webových stránek v praxi. 1. vyd. Praha: Grada, 2009, 516 s. ISBN 9788025141632

[15] LAND, Kenney. Onscreen controls 8 styles [online]. 2014. vyd. [cit. 2015-03-09]. Dostupné z: http://opengameart.org/content/onscreen-controls-8-styles

[16] MCLEOD, Aron. GITHUB. MelonJS Wiki [online]. 2015. vyd. [cit. 2015-03-09]. Dostupné z: https://github.com/melonjs/melonJS/wiki

[17] MCLEOD, Aron. GITHUB. MelonJS boilerplate [online]. 2015. vyd. [cit. 2015-03-09]. Dostupné z: https://github.com/melonjs/boilerplate

[18] MUSIC TECHNOLOGY GROUP. Freesound [online]. 2005. vyd. [cit. 2015-03-09]. Dostupné z: https://www.freesound.org

[19] ADOBE. Brackets [online]. 2014. vyd. [cit. 2015-03-09]. Dostupné z: http://brackets.io

[20] About us: HTML5 game development. [online]. [cit. 2015-03-19]. Dostupné z: http://www.photonstorm.com/html5

[21] JS requestAnimationFrame: za lepší vykreslování. ITNETWORK. [online]. [cit. 2015-03-19]. Dostupné z: http://www.itnetwork.cz/tutorial-javascript-requestanimationframe-za-lepsi-vykreslovani

[22] Rendering shapes: using MelonJS. AGMPROJECTS. [online]. [cit. 2015-03-19]. Dostupné z: http://agmprojects.com/blog/rendering-shapes-and-drawings-via-the-canvas-in-melonjs

[23] Implementing Particles with melonJS. SCHMITZ, Andre Antonio. CIANGAMES. [online]. 2014. vyd. [cit. 2015-03-27]. Dostupné z: https://www.packtpub.com/books/content/implementing-particles-melonjs

# Seznam obrázků

# Seznam tabulek