STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
ZOCHOVA 9, BRATISLAVA

2D Videohra

Meno kandidáta: Samuel Chinonso John

Trieda: IV. D

Vedúci práce: Bc. Lenka Ráčková

Šk. rok: 2024/2025

STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
ZOCHOVA 9, BRATISLAVA

2D Videohra

Meno kandidáta: Samuel Chinonso John **Trieda:** IV. D

Vedúci práce: Bc. Lenka Ráčková

Rozsah: 27 strán, 24 obrázkov, 0 tabuliek, 1 príloh

Odovzdané: **Podpis vedúceho práce:**

Čestné prehlásenie

Čestne prehlasujem, že problematiku týkajúcu sa náplne zadaného projektu v rámci praktickej časti odbornej zložky maturitnej skúšky formou obhajoby som spracoval sám za pomoci uvedenej použitej literatúry. Inú, ako uvedenú literatúru som nepoužil.

V Bratislave dňa: Samuel Chinonso John

Poďakovanie

Rád by som úprimne poďakoval svojej vedúcej práce za možnosť pracovať pod jej vedením a za cenné rady, trpezlivosť a podporu, ktorú mi poskytovala počas celej tvorby maturitného projektu. Veľmi si vážim jej ochotu, ako aj jej povzbudenie, ktoré mi pomohlo prekonať výzvy, s ktorými som sa stretol. Taktiež by som rád poďakoval všetkým spolužiakom a kamarátom, ktorí sa podieľali na testovaní nášho maturitného projektu. Ich ochota, spätná väzba a cenné postrehy mi pomohli vylepšiť a doladiť moju prácu. Nakoniec by som chcel podakovat aj kamaratovy Fin „WeeWooX“ za podiele na vytvorení niektorých zvukových efektoch pre projekt.

OBSAH

ZOZNAM OBRÁZKOV

Elektronické prílohy

Elektronická príloha č. 1: Dostupná stiahnutelná hra na stránke itch.io: https://croxwitsox.itch.io/timeless

Úvod

Tvorba hier nás vždy fascinovala, pretože kombinuje kreativitu, technické zručnosti a logické myslenie. Baví nás navrhovať herné mechaniky, vytvárať interaktívne prostredia a sledovať, ako sa naše nápady premieňajú na funkčnú hru. Práve preto sme si zvolili vývoj hry ako tému našej maturitnej práce. Je to oblasť, v ktorej sa cítime prirodzene, radi sa v nej zdokonaľujeme a vidíme v nej svoju budúcnosť. Táto práca nám zároveň poskytla príležitosť prehĺbiť naše znalosti v programovaní a dizajne hier.

Cieľom tejto práce bolo vytvoriť kompletnú 2D hru v programe *GameMaker Studio* (softvér na vývoj hier), ktorý využíva vlastný programovací jazyk GML. Hra bude zameraná na pohyblivosť hráča, čo znamená, že hráč bude odmenený za presné ovládanie a pochopenie jednotlivých herných prvkov.

Hra bude celkovo primerane náročná, dostatočne prístupná pre menej pokročilých hráčov, aby si ju mohli užiť bez frustrácie, no zároveň ponúkne výzvu pre skúsenejších hráčov. Herné prvky budú navrhnuté tak, aby sa ich hráč mohol prirodzene naučiť a postupne ich kombinovať. S rastúcou náročnosťou bude hra testovať jeho schopnosti a poskytne priestor na neustále zlepšovanie. Tí, ktorí budú hľadať väčšiu výzvu, v nej nájdu vysoký strop pre zdokonaľovanie svojich zručností.

Dúfame, že konečný výsledok zabaví hráča aj v prípade, že nemá veľa skúseností s plošinovkami. Zároveň veríme, že hra zaujme aj náročnejších hráčov a ponúkne im možnosť zlepšovať sa a posúvať svoje limity.

TEORETICKÁ ČASŤ

1 Žáner

Inšpiráciou pre našu hru sú populárne tituly, ako napríklad Hollow Knight, Celeste či Vector. Hra sa zároveň výrazne inšpiruje titulom *Ultrakill*, ktorého silnou stránkou sú prepracované herné mechaniky. Tie sú na základnej úrovni jednoduché a prístupné, no pri dostatočnej investícii času umožňujú hráčovi dosiahnuť neuveriteľnú úroveň ovládania a kreativity pri ich využívaní. Tento princíp sme sa snažili implementovať aj do nášho projektu.

Obrázok 1 – Príklad hry v štýle "*metroidvanie*" *Hollow Knight*

Metroidvania je názov odvodený od populárnych hier *Metroid* a *Castlevania*, ktoré sú najväčšími predstaviteľmi tohto žánru. Tento štýl opustil klasický lineárny dizajn úrovní bežný v platformových hrách a namiesto toho vytvoril jeden veľký prepojený svet, ktorý hráč môže preskúmavať. Zároveň nahradil tradičné požiadavky platformových hier, ako precízne skákanie a pohyb, a zameral sa viac na objavovanie sveta a boj s unikátnymi nepriateľmi.

Metroidvanie často obsahujú cestičky, ktoré sú na začiatku neprístupné, až kým hráč nepreskúma ďalšie časti sveta a nevráti sa k nim neskôr. Hra čerpá najmä z *metroidvania* štýlu, predovšetkým v dizajne úrovní a prepojeného herného sveta, kde hráč objavuje nové cesty a postupne nadobúda nové schopnosti.

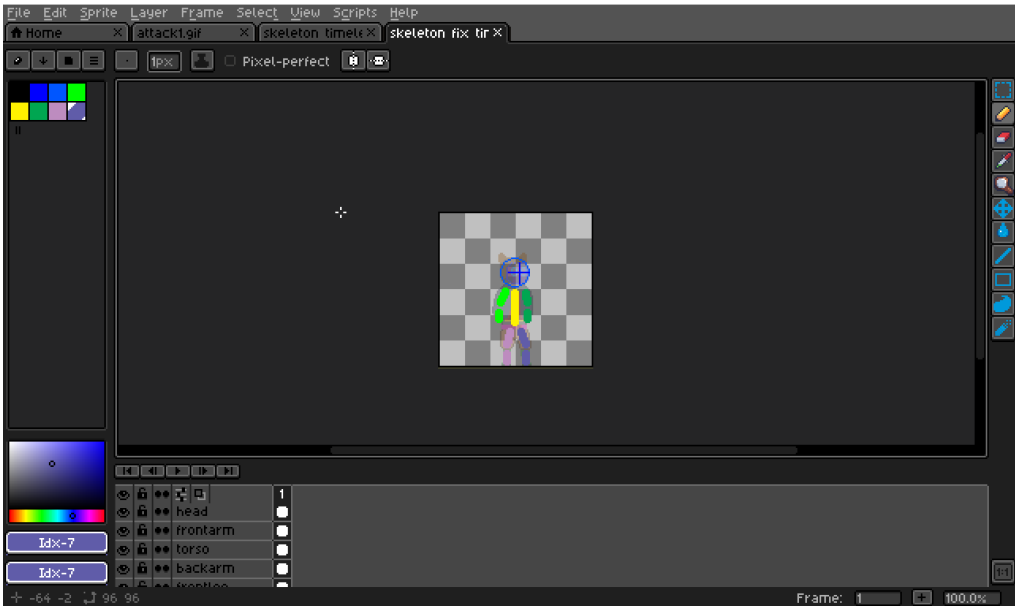
Táto hra čerpá aj zo štýlu klasických plošinových skákačiek, pričom kladieme dôraz na dynamický pohyb a precízne ovládanie. Spájame tak plynulé preskúmavanie Obrázok

Grafika

Grafická stránka projektu je vytvorená v štýle pixel art, čo je forma digitálneho umenia, kde sú obrazy zložené z jednotlivých pixelov. Každý pixel predstavuje základný stavebný prvok, z ktorého vznikajú tvary, postavy a prostredia. Tento štýl dodáva hre charakteristický retro vzhľad, pripomínajúci klasické videohry, no zároveň umožňuje detailnú a precíznu vizuálnu prezentáciu.

1.1 Grafický nástroj

Na tvorbu pixel artovej grafiky používame program Aseprite, ktorý je špeciálne navrhnutý pre tvorcov pixel artu. Ponúka množstvo nástrojov a funkcií, ktoré uľahčujú prácu s jednotlivými pixelmi, animáciami a detailnými grafikami, čím výrazne prispieva k efektívnemu a precíznemu vizuálnemu spracovaniu hry.

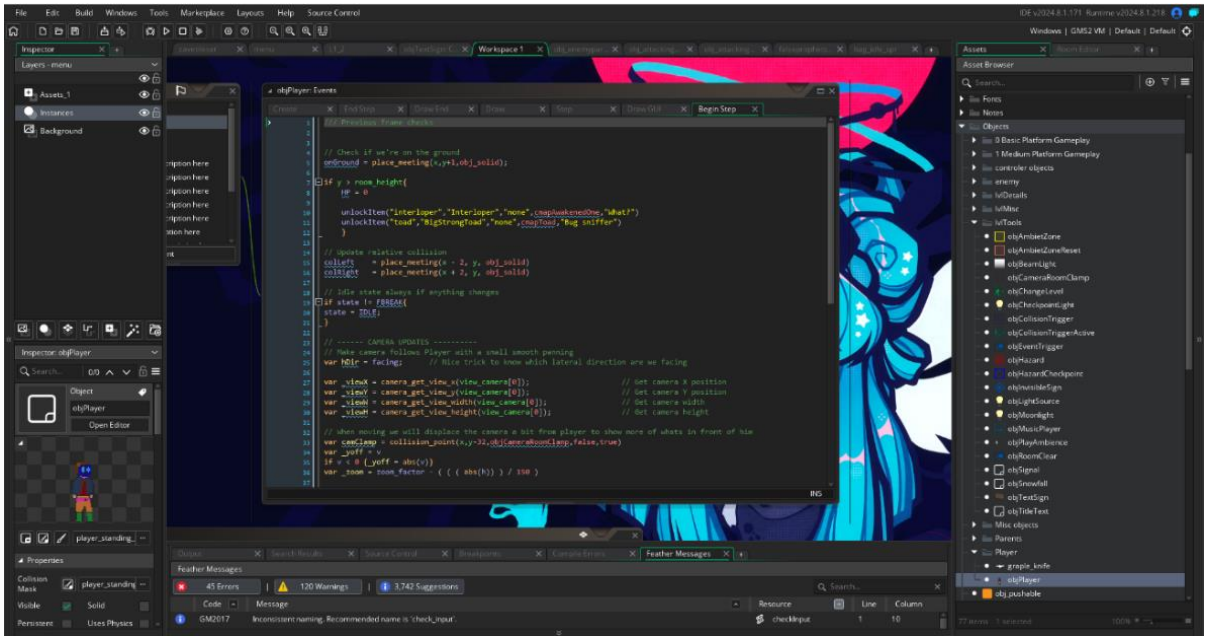


Obrázok 2 – Pracovný priestor v programe Aseprite

2 Vývojový nástroj

GameMaker Studio je herný engine od spoločnosti YoYo Games, ktorý sa špecializuje na vývoj 2D hier. Hoci má určité obmedzenia, ako napríklad neschopnosť vytvárať 3D hry, pre náš projekt to nepredstavuje problém. Keďže sa zameriavame na 2D hru, GameMaker Studio je pre nás ideálnym nástrojom, pretože spĺňa všetky požiadavky potrebné na efektívny vývoj.

GameMaker v porovnaní s inými vývojovými nástrojmi využíva koncept eventov (kroký ktore podla poradia spustia kod), ktoré umožňujú unikátne spracovanie kódu. Tieto eventy poskytujú flexibilitu pri organizovaní a správnom vykonávaní rôznych činností v hre. Napríklad, pri step evente máme začiatok a koniec kroku, pričom každý objekt najprv vykoná začínajúci kód (begin) a následne končiaci kód (end). Tento prístup nám umožňuje vykonávať medzikalkulácie a uchovávať dáta medzi rôznymi objektami. V podstate nahrádza asynchrónne spracovanie, pretože jednotlivé kroky sú riadené v jasne definovanom poradí. Týmto spôsobom je možné dosiahnuť lepšiu kontrolu nad prúdom hry a efektívnejšie spravovať zložitejšie interakcie medzi objektami.



Obrázok 3 - Pracovný priestor v programe Gamemaker studio

Vybrali sme GameMaker Studio, pretože máme s ním skúsenosti a považujeme ho za veľmi intuitívny nástroj na tvorbu 2D hier. Tento engine ponúka základný sprite editor, ktorý je veľmi užitočný pri tvorbe grafiky a animácií. Okrem toho využíva vlastný programovací jazyk GML (GameMakerov programovací jazyk), ktorý je jednoduchý na učenie a poskytuje dostatočnú flexibilitu pre tvorbu herných mechaník.

3 Herný dizajn

3.1 Spokojnosť hráča

Ak chceme urobiť dobrú hru musí byť motivujúca a poskytovať hráčovi dostatočný dôvod pokračovať v jeho hraní. Je dôležité nájsť správne vyváženie medzi rastúcou obtiažnosťou a odmenami pre hráča. Postupné zvyšovanie náročnosti by malo byť dostatočne plynulé, aby hráč nepociťoval frustráciu, no zároveň mal dostatok výziev a bol odmenený za prekonanie prekážok. Odmeny v podobe nových schopností, predmetov alebo prístupu k novým oblastiam by mali hráča motivovať, aby sa neustále zlepšoval a hľadal spôsoby, ako zdolať ďalšie výzvy.

3.2 Prostredie

Prostredie by malo prirodzene viesť hráča a zároveň zachovať vizuálnu prehľadnosť. V plošinovkách je dôležité, aby dizajn neprekryl kľúčové prvky, ako sú postava či prekážky. Preto je vhodné vyhnúť sa príliš detailným pozadiam, ktoré by mohli odvádzať pozornosť. Zároveň však musí byť prostredie vizuálne príťažlivé a podporovať atmosféru hry. Správna rovnováha medzi jednoduchosťou a estetikou pomáha vytvoriť pútavý a intuitívny herný zážitok.

PRAKTICKÁ ČASŤ

4 Grafické implikácie

Grafická stránka hry je kľúčovým prvkom, ktorý ovplyvňuje celkový zážitok hráča. Grafický štýl kombinuje vibrantnú atmosféru hry Hollow Knight s typickými sci-fi rastlinami. Zároveň pridáva aj temnotu hororových hier v štýle VHS.

4.1 Vytváranie kostýmov

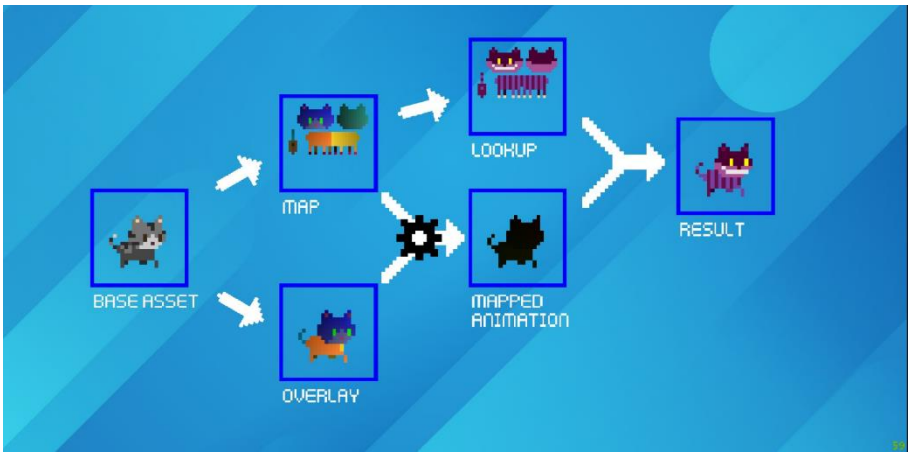
Pre odmenenie hráča projekt obsahuje zberateľné predmety, tzv. *collectibles*, ktoré sa bežne vyskytujú v iných hrách ako odmeny pre hráčov, ktorí si dávajú čas na preskúmanie každého rohu herného sveta. V tomto projekte sú collectibles vo forme kostýmov pre hráča.

Nápadom bolo vytvoriť jeden predvolený vizuál postavy a následne pomocou kódu dynamicky meniť farby jednotlivých pixelov v závislosti od vybraného kostýmu. Tento prístup umožňuje efektívne a flexibilne upravovať vzhľad postavy bez potreby vytvárania viacerých grafických verzií.

4.1.1 Generácia 1 – UV maping

Prvá generácia tohto systému využívala techniku UV *mapping*. UV *mapping* je proces, pri ktorom sa 2D textúra „natiahne“ na 3D model v hrách a počítačovej grafike. Osy “U“ a „V“ predstavujú šírku a výšku textúry (namiesto klasických “X“ a “Y“). Cieľom UV mapovania je zabezpečiť, aby sa textúry, ako napríklad povrchy kože, látky alebo detaily objektov, presne prispôsobili povrchu 3D modelu.

UV mapping sa dá využiť aj v 2D hrách, kde preskočíme proces „natiahnutia“ 2D textúry a rovno ju mapujeme na 2D postavičku. V našom prípade sme namiesto klasických farieb na postavičke vymenili každý pixel za unikátnu farbu, na ktorú sme následne namapovali inú farbu.



Obrázok 4 – Vizualizácia techniky UV Maping [1]

Na obrázku vidíme vizualizáciu procesu UV mappingu. Naľavo je postavička, ktorú chceme mapovať. Pod postavičkou je overlay, kde sme každému pixelu priradili unikátnu farbu. Nad týmto overlay sa nachádza mapa, ktorá ukazuje, kde sa nachádza každá farba, a na záver je lookup, ktorý určuje, ako bude vyzerať konečný výsledok.

Keď máme všetky tieto dáta, vytvoríme skript, ktorý načíta všetky pixely na overlay. Tento skript zistí, kde sa dané farby nachádzajú na mape, následne získa farbu z príslušnej pozície v lookup obrázku a túto farbu aplikuje na pôvodnú pozíciu v overlay obrázku.

Tento proces je komplikovaný a časovo náročný, pretože je potrebné každému pixelu priradiť unikátnu farbu pre každý snímok animácie, vrátane skrytých častí postavy, ktoré sa objavia len pri animácii (napr. chrbát). Preto sa táto technika nehodí pre malé a jednoduché obrázky, ako je pixel art, a bolo potrebné nájsť nový prístup.

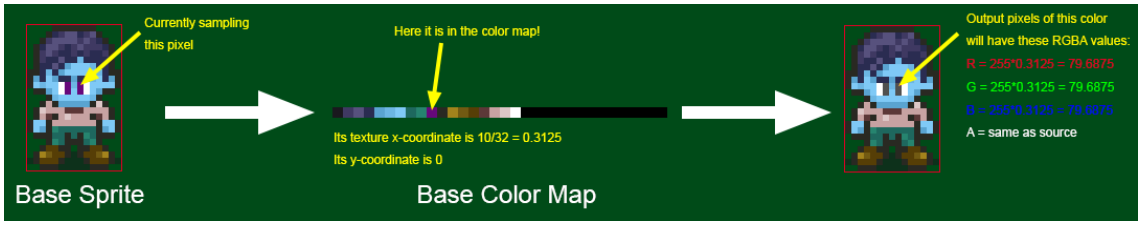
4.1.2 Generácia 2 – Color maping

V druhej verzii sme využili metódu color mapping, ktorá namiesto toho, aby každému pixelu priradila unikátnu hodnotu, indexuje jednotlivé časti postavy pomocou špecifických farieb. Napríklad vlasy sú modré, maska tmavomodrá, oči zelené a podobne. V hre následne pomocou skriptu zmeníme farby pixelov, podobne ako pri UV mappingu, ale namiesto komplexnej farebnej mapy nám stačí jednoduchá farebná paleta. Tento prístup zjednodušuje proces zmeny farieb a znižuje nároky na plynulý vývoj.

Najprv pre každý blok farby, ktorý chceme ovládať, zakreslíme na postavičke unikátnu farbu a vytvoríme si základnú paletu, ktorá bude jednoduchý obrázok s rozmermi 32x1 pixelu. Táto paleta bude slúžiť ako základ, ktorý bude určovať poradie našich farieb.

V hre prejdeme všetky pixely vykreslenej postavičky a porovnáme ich s našou 32x1 paletou. Keď nájdeme zhodu, zistíme horizontálnu pozíciu tejto farby na palete. Napríklad, ak je pozícia farby na palete 10, nahradíme ju monochromatickou farbou, kde všetky hodnoty RGB upravíme na 10/32, čím ich indexujeme. Naša nová farba bude teda mať hodnoty: r = 0.3125, g = 0.3125, b = 0.3125 alebo teda index 10 [2].

Ďalej použijeme *shader*, ktorý vezme monochromatickú postavičku a prejde každý pixel. Pre každý pixel vyberieme hodnotu z RGB (ktorá je rovnaká pre všetky kanály) a vynásobíme ju veľkosťou palety. Týmto získame pozíciu farby na palete, z ktorej vykreslíme novú farbu. Potom sa pozrieme na pozíciu na novej palete, ktorú chceme použiť na prekreslenie postavičky, a daný pixel zakreslíme.



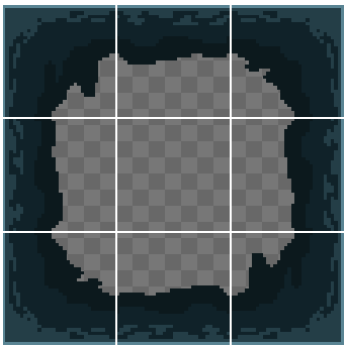
Obrázok 5 – Vizualizácia techniky Color maping [2]

Tento skript využívame aj na vykreslenie životov na obrazovke, konkrétne v ľavom hornom rohu.

Obrázok 6 - priklad pouzitia palety "*Karlson*" v projekte

4.2 Tileset

Pri dekorovaní miestností využívame systém *tileset*. *Tileset* je kolekcia rastrových obrázkov usporiadaných v mriežke, kde každý „grid cell" reprezentuje jeden skladačkový blok. Z týchto usporiadaných blokov môžeme vytvoriť celý svet, čím sa vyhneme potrebe kresliť unikátne časti každej miestnosti. *Tilesety* môžu mať rôzne veľkosti, no v projekte nám postačuje 32x32 *tileset*. Tento prístup využívame najmä na ušetrenie času, pretože umožňuje vytvoriť modulárny základ pre celý herný svet.



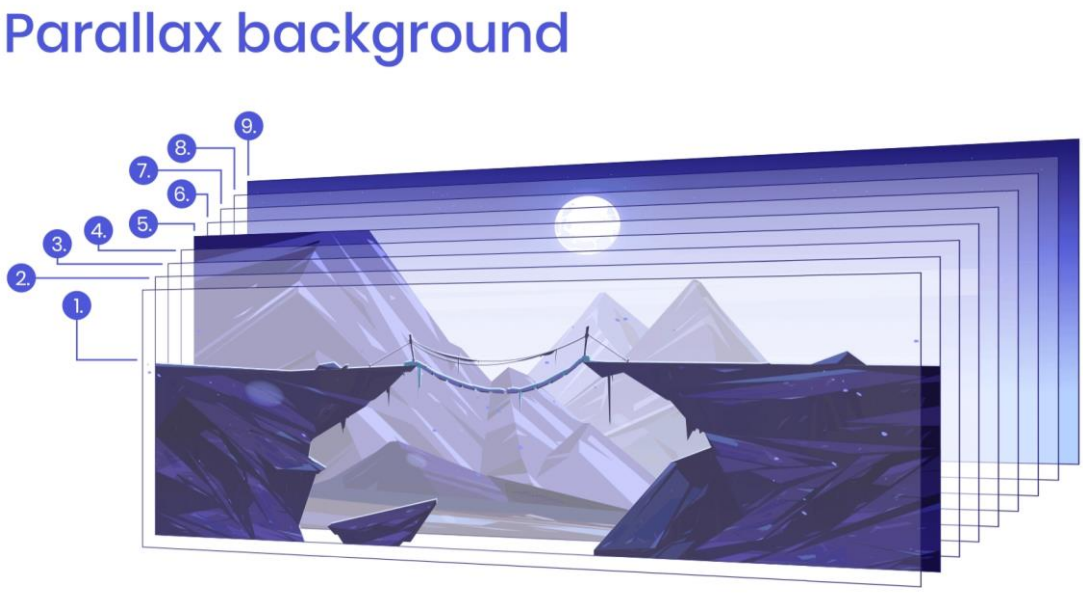
Obrázok 7 – Kúsok tilesetu použitý v hre

4.3 Grafické efekty

V projekte sme využili niekoľko grafických efektov, ktoré zlepšujú herný zážitok a vizuálnu príťažlivosť. Tieto efekty sa hlavne podieľajú na vytvorení atmosféry, ale tiež zlepšujú interaktivitu a zrozumiteľnosť herného sveta.

4.4 Parallax

Pozadia v hrách často vykresľujeme ako jednoliatu farbu alebo gradient. V mnohých prípadoch to úplne postačuje, pretože ide o minimálny štýl, ktorý neodvádza pozornosť hráča, a s farbou sa ďalej nepracuje, keďže všade vyzerá rovnako. Pre tento projekt sme sa však rozhodli vytvoriť pozadie, ktoré dopĺňa scenár a vnáša viac života do prázdnych miestností. Na dynamické pozadie sme využili techniku *parallax*. *Parallaxa* je vizuálny efekt, pri ktorom vykresľujeme viacero obrázkov na sebe a pohybujeme ich podľa „blízkosti“. Týmto spôsobom simulujeme 3D hĺbku, aj keď pracujeme s 2D plochou.



Obrázok 8 – Vizualizácia techniky Parallax [3]

4.5 Osvetlenie

Osvetlenie pridáva hrám veľké čaro a dokáže výrazne ovplyvniť atmosféru. Niekedy stačí len správna kombinácia farieb a osvetlenia, aby hra vyzerala atraktívne. V tomto projekte sme sa rozhodli neimplementovať komplikovaný osvetľovací systém, keďže GameMaker nemá zabudovanú podporu pre osvetlenie. Vytvorenie vlastného systému by si vyžadovalo veľa času a mohlo by byť náročné na výkon, najmä pri veľkých miestnostiach, čo by mohlo viesť k vysokému zaťaženiu počítača. Preto sme zvolili jednoduchý spôsob simulácie svetla, ktorý bol efektívny a zároveň neovplyvnil výkon hry.

Najprv vykreslíme na kameru plnú čiernu farbu s priesvitnosťou, čím zafarbíme všetko na tmavo a v podstate prekryjeme celú obrazovku temnotou. Následne, pre každý objekt osvetlenia, vytvoríme dieru v tejto priesvitnej čiernej vrstve pomocou zabudovanej funkcie *blendmode* v GameMakeri. Táto funkcia nám umožňuje odpočítať hodnotu priesvitnosti z pixelov čiernej farby pomocou iných obrázkov. To znamená, že ak vykreslíme plný biely kruh na čierne pozadie, výsledok bude "diera", kde bude zobrazená len priesvitná časť, a nie samotný biely kruh. Takto vytvoríme efekt falošného svetla, alebo skôr absenciu tmy. Na záver, ak chceme pridať farbu do svetla, jednoducho nakreslíme rovnaký kruh, tentokrát už s farbou, priamo na vyrezanú dieru. Týmto spôsobom dosiahneme efekt osvetlenia, ktorý je jednoduchý a efektívny, ale stále pôsobivý.

4.6 Extra efekty

V projekte sú taktiež využité particles, ktoré používame pre extra efekty a pridávanie viac života. Particles sa používajú v prípadoch, kedy chceme mimikovať animáciu alebo efekt, napríklad prach na zemi. Prach by sa mohol teoreticky naanimovať, no nevyzeralo by to prirodzene. Namiesto toho vieme vytvoriť jednoduchý obraz vznietajúceho sa prachu a pomocou particle systému dynamicky randomizovať veľkosť, rýchlosť, vzdialenosť a podobne. Toto by bez particle systému stálo oveľa viac času, keďže by sme vyžadovali rôzne animácie podľa vzdialenosti a rýchlosti. Particles vieme využiť aj v jednoduchších prípadoch, napríklad sneh, ktorý by sa animáciami veľmi zle kreslil.

5 Funkcionálne implikacie

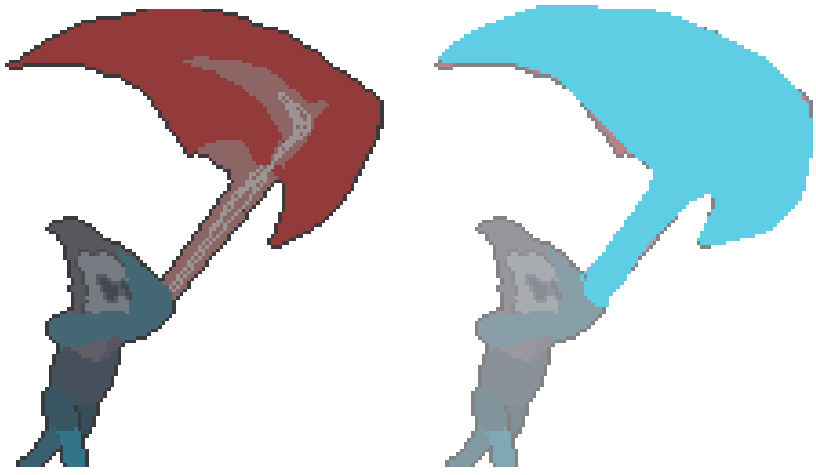
Výzvy a funkcie implikované v projekte.

5.1 Hitbox a Hurtbox

V našej hre je veľmi dôležité správne oddeliť *hitbox* a *hurtbox* pre útoky, aby sme dosiahli presné detekcie kolízií. ***Hitbox*** je oblasť, ktorá určuje, kde a ako sa útok môže uskutočniť, či už ide o útok hráča, nepriateľa alebo iný mechanizmus v hre. Tento *hitbox* je oddelený od animácie, čo znamená, že sa nachádza okolo útoku a určuje, kde sa bude kolízia s cieľom vykonávať.

Na druhej strane, ***hurtbox*** označuje oblasť, ktorá reprezentuje, kde môže postava utrpieť poškodenie, to znamená, kde sa nachádza postava, ktorá môže byť zasiahnutá útokom. Tento hurtbox sa zvyčajne prispôsobuje podľa aktuálnej pozície a animácie postavy, čo znamená, že môže meniť svoju veľkosť a tvar v závislosti od akcie, ktorú postava vykonáva.

Oddelením hitboxu a hurtboxu môžeme zabezpečiť presnejšie a flexibilnejšie správanie v boji, kde sa útoky a zranenia detegujú nezávisle od seba, čo zvyšuje dynamiku a kontrolu nad hrou.



Obrázok 9 – Porovnanie snímok postavičky a jej hitbox

6 Uchovávanie hry

6.1 Uchovávanie stavu

V projekte uchovávame stav hry, čo je vizuálne zobrazené v hre správou „gamestate saved“. Pri ukladaní však neuchovávame všetky objekty a každu premenu, ale len tie, ktoré môžu zmeniť svoj stav. Na dosiahnutie tohto sme vytvorili skript, ktorý prejde všetky objekty, ktoré chceme uložiť, najmä hráča, nepriateľov a podobne. Potom uložíme potrebné informácie, ako sú ich súradnice a ich stav, do knižníc, ktoré následne zapisujeme do JSON súboru, v zložke „appdata/Local/Timeless/game_save.json“. Každýkrát, keď chceme načítať stav hry (napríklad po načítaní checkpointu), jednoducho prečítame všetky informácie z týchto knižníc a načítame zodpovedajúce premenné do hry.

6.2 Uchovávanie kostýmov

Ukladanie kostýmov je spracované v samostatnom JSON súbore v tej istej zložke. Ukladáme tam všetky kostýmy, ktoré hráč získal, a každý kostým je referencovaný pomocou kľúčov. K týmto kľúčom priraďujeme aj farebnú paletu ktorá patrí ku kluču.

6.3 Uchovávanie nastavení

Nastavenia sa ukladajú v tej istej zložke, avšak tentokrát používame zabudovaný formát súboru .ini, ktorý slúži ako konfiguračný súbor. Tento formát funguje podobne ako knižnica a umožňuje nám jednoducho ukladať a načítavať rôzne nastavenia hry.

7 Skripty

Viaceré skripty boli vytvorené pre hru, ktoré fungovali ako akési frameworky. Z týchto frameworkov sme následne budovali ďalšie funkcie a herné mechaniky. Skripty nám umožnili vytvoriť základné štruktúry, na ktorých sme mohli ďalej stavať, a tým ušetriť čas pri implementácii opakujúcich sa funkcií. Tento prístup tiež zjednodušil úpravy a rozšírenia hry v priebehu vývoja, keďže akákoľvek zmena v základnom frameworku sa automaticky prejavila vo všetkých častiach hry, ktoré tento kód využívali.

7.1 Pohyb

Základom hry je pohyb, a preto je dôležité zabezpečiť, aby pohyb hráča, postáv a iných predmetov bol plynulý, kontrolovateľný a responzívny. Aby sme to dosiahli, potrebujeme simulovať jednoduchú fyziku, ktorú následne môžeme voľne kontrolovať. Každému objektu, ktorý sa pohybuje, priraďujeme základné fyzikálne výpočty, ako sú horizontálna a vertikálna rýchlosť, gravitačné sily, kolízie a podobne.

Týmto spôsobom môžeme presne riadiť pohyb a interakcie objektov, pričom ich správanie zostáva predvídateľné a kontrolovateľné, čo je kľúčové pre herný zážitok pri plošinových hrách.

Kód je rozdelený do fáz, ktoré bežia pri každom snímku hry, čo umožňuje vykonávanie potrebných výpočtov a kalkulácií ovplyvnených interakciami s ostatnými objektmi.

V prvej fáze definujeme niekoľko premenných, ktoré uchovávajú základné podmienky. Hlavne zisťujeme, či sa pod hráčom nachádza zem, teda či hráč stojí na zemi, a tiež či sa dotýka stien naľavo alebo napravo.

V ďalšej fáze, podľa definovaných podmienok a správania objektov, aktualizujeme rýchlosť pohybu. Na základe vstupov od hráča a interakcií s prostredím, ako sú kolízie so stenami alebo s podlahou, priraďujeme objektu novú hodnotu kalkulovanú rýchlosti. V tejto fáze sú tiež zohľadnené ďalšie faktory, ako je gravitácia alebo zrážky, ktoré môžu ovplyvniť pohyb objektu a prispieť k plynulosti jeho animácie.

Vo viacerých funkciách v tejto medzi fáze využívame lineárnu interpoláciu, čo je kľúčové pre plynulé riadenie pohybu. Významne sa používa napríklad vo funkciách pre pohyb doľava a doprava. Lineárna interpolácia nám pomáha dosiahnuť hladké zrýchlenie, keď sa hráčova rýchlosť zvyšuje od nuly až po maximálnu hodnotu.

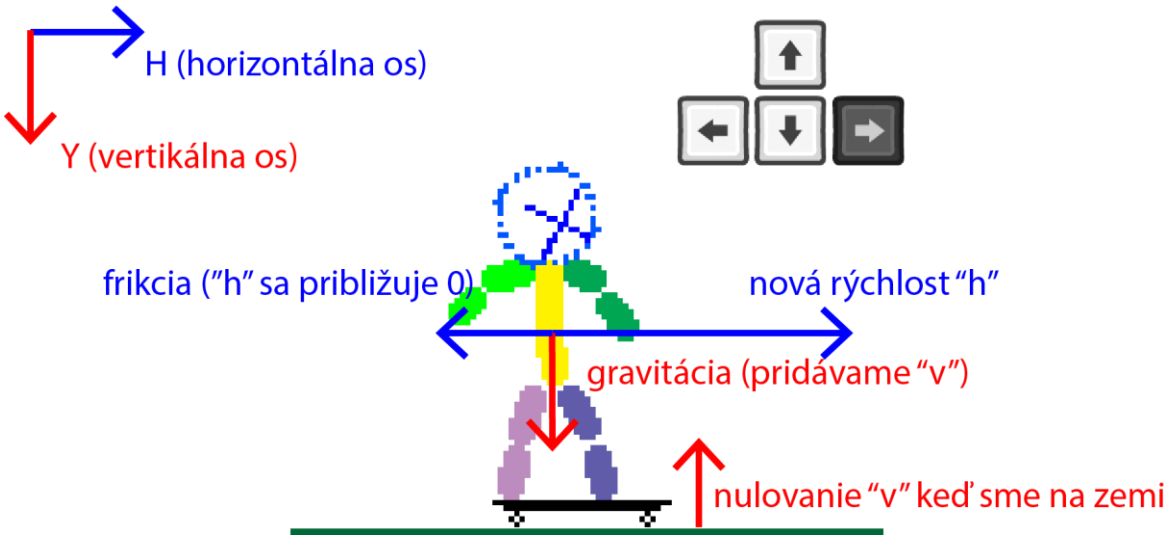
Vzorec na výpočet používa aktuálnu horizontálnu rýchlosť, zistí rozdiel medzi aktuálnou rýchlosťou a maximálnou rýchlosťou, a potom vypočíta novú hodnotu rýchlosti na základe akcelerácie a percentuálnej hodnoty z tohto rozdielu.

Príklad: Ak máme horizontálnu rýchlosť **h** = 0, maximálnu rýchlosť **maxH** = 10 a akceleráciu **Acel** = 0.5, nový výpočet rýchlosti bude vyzerať takto: nové h = 0.5 * (rozdiel medzi maximálnou rýchlosťou a aktuálnou rýchlosťou) V tomto prípade je rozdiel medzi maximálnou rýchlosťou a aktuálnou rýchlosťou 10 - 0 = 10, takže výsledná rýchlosť nové h = 0.5 * 10 = 5. Tento proces zaručuje plynulé prirodzené zrýchlenie.

Následne, v poslednej fáze, kde už máme vypočítané rýchlosti a informácie o pohybe, stačí len aplikovať hodnoty našej novej rýchlosti na pozíciu hráča.

Tieto kroky nám umožňujú riadiť všetky základné pohyby v hre. Rôzne druhy pohybov sa riadia pomocou špecifických premenných, ktoré kontrolujú dynamiku pohybu, ako aj faktory, ktoré ovplyvňujú zmenu rýchlosti a smeru. Každý pohyb je postupne prispôsobovaný na základe aktuálnych podmienok, čím sa dosahuje plynulý pohyb hráča v hre.

Po dokončení všetkých funkcií pohybu bola vytvorená aj zjednodušená verzia pre nepriateľov, ktorá obsahovala menej parametrov, ktorá funguje rovnako.



Obrázok 10 – Zjednoduchčené znázornenie aplikovanej rýchlosti v hre

7.2 State machine

Hráč má viacero animácií a možností pohybu, čo znamená, že neustále mení správanie a vzhľad. Preto sme na uľahčenie správania naprogramovali *state machine* (spracovávanie stavov), ktorý nám umožňuje definovať správanie v každom stave. Zadefinovali sme stavy ako čísla, napríklad ATTACK = 10. Tým pádom, keď v kóde odkazujeme na ATTACK, v skutočnosti odkazujeme na číslo 10, čo uľahčuje manipuláciu a výpis v *state machine*.

V *step evente* najprv resetujeme náš stav, keďže vždy, keď nie sme v žiadnom stave, chceme byť v základnom stave, čo v našom prípade znamená stavu na **IDLE**. Ďalej cez kód vyberieme, aký chceme nový stav, ak v celom kóde nie je splnená žiadna podmienka, ktorá by zmenila stav, zostáva stav **IDLE**.

Nakoniec, v poslednej fáze kódu pred vykreslením snímky môžeme podrobnejšie definovať samostatné správanie pre každý stav. Toto je „srdce“ nášho stroja stavov, tu reálne ovládame, čo každý stav robí, a to pomocou jednej podmienky typu switch-case.

V niektorých prípadoch používame aj „***substate***“ (podstav). Pri niektorých stavoch je potrebné mať viacero správaní. Napríklad v stave RUN, ktorý nastane, keď hráč stláča šípku doprava alebo doľava. Ak hráč ešte stále udržiava rýchlosť v opačnom smere, mala by sa prehrávať animácia brzdenia. Tento stav RUN teda vlastne obsahuje dve rôzne správania – brzdenie a plynulé bežanie. Takéto správanie sme riešili pomocou *substatu*.

Substate je iba jednoduchý kľúč, ktorý môže byť čokoľvek, čo si zapamätáme. V stave následne vyhľadáme náš kľúč pomocou **switch** podmienky a definujeme správanie. Na rozdiel od **stavu** nemusíme ***substate*** resetovať, keďže ide o unikátny kľúč. To znamená, že aj keď ho nevynulujeme a dostane sa do iného stavu, než v ktorom mal byť, nový stav ho nerozpozná a nič s ním neurobí.

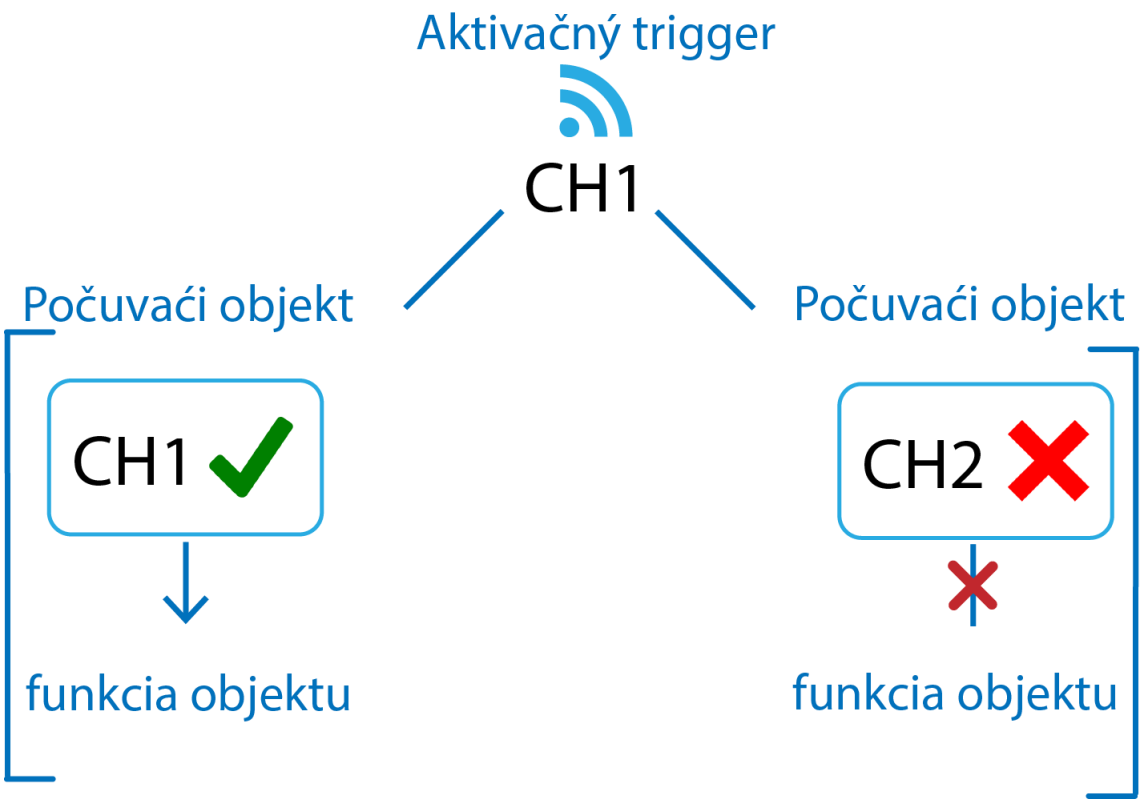
7.3 Trigger system

Pri tvorbe sveta sme sa stretli s viacerými situáciami, keď bolo potrebné ovplyvniť ďalšie objekty. Často, napríklad pri takzvaných "*scripted events"*, kedy chceme špecificky spustiť, vytvoriť alebo aktivovať nejaký objekt, musíme naprogramovať podmienky. Tento prístup však môže rýchlo viesť k neprehľadnosti v kóde, najmä ak chceme spustiť viacero udalostí v presne definovanom čase.

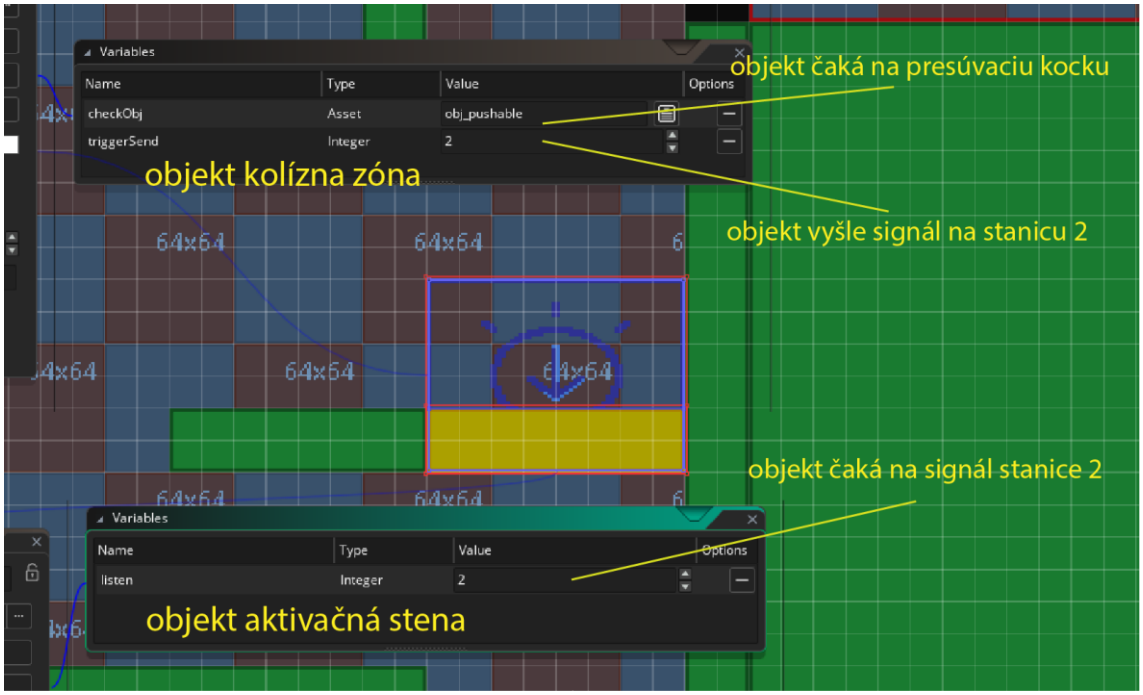
Z tohto dôvodu sme do objektov implementovali *trigger* systém. Tento systém umožňuje objektom v hre vzájomne komunikovať prostredníctvom takzvaných kanálov. Aktivujúci *trigger* vyšle signál na konkrétny kanál, ktorý následne rozpošle tento signál všetkým objektom, ktoré tento kanál "počúvajú". Ak objekt počúva kanál, ktorý sa zhoduje so signálom, tento signál prijme a na jeho základe môže spustiť príslušnú funkciu.

Pri tvorbe sveta bolo viacero prípadov, keď sme chceli ovplyvniť ďalšie objekty. Často, napríklad pri takzvaných „*scripted events*", keď chceme špecificky spustiť, vytvoriť alebo aktivovať nejaký objekt, musíme naprogramovať podmienku. To však môže veľmi rýchlo spôsobiť neprehľadnosť v kóde, najmä ak chceme spustiť veľa udalostí v presne definovanom čase.

Z tohto dôvodu sme do objektov naprogramovali *trigger* systém. Vďaka nemu môžu objekty v hre medzi sebou komunikovať prostredníctvom takzvaných kanálov. Aktivujúci *trigger* vyšle signál na konkrétny kanál, ktorý ho následne rozpošle všetkým objektom, ktoré tento kanál "počúvajú". Ak objekt počúva kanál, ktorý sa zhoduje so signálom, tento signál prijme a na jeho základe môže spustiť skript.



Obrázok 11 – Vizualizácia funkcie “Trigger system“



Obrázok 12 – Príklad aplikovania funkcie “Trigger system”

7.4 Kamera

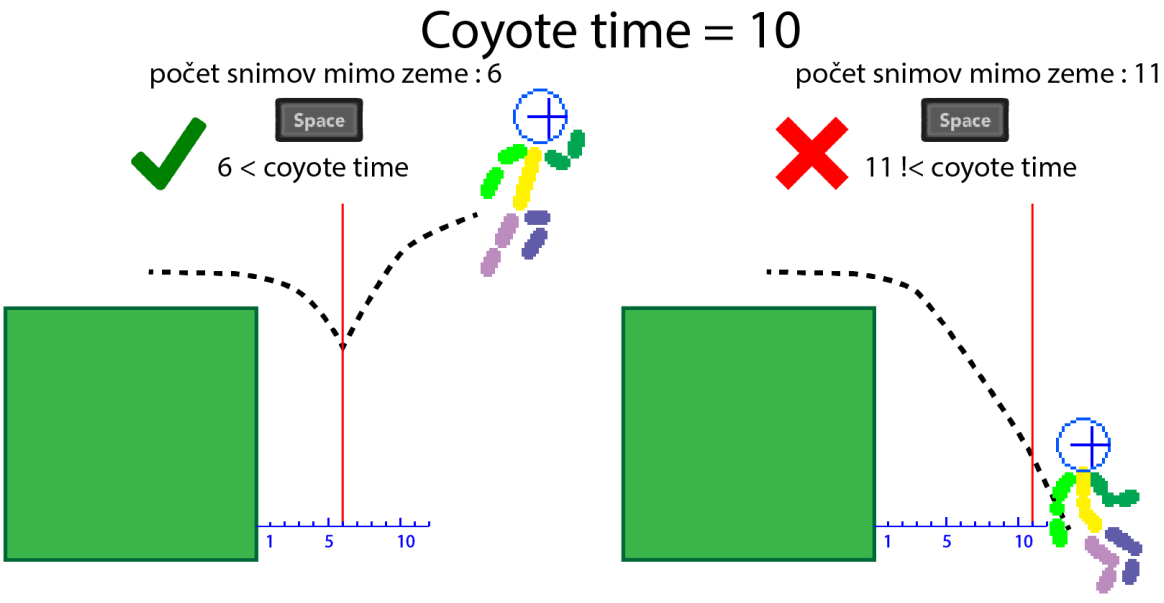
V Gamemakery je zabudvaný kamerový systém ten však nebol dostačujúci. Rozhodli sme sa teda urobit jednoduchý vlastný kamerový systém, hlavne aby sme mohli volne využiť lineárnu interpoláciu

8 Qol

V herných projektoch a hrách celkovo sú často implementované takzvané „quality of life“ (QoL) vylepšenia, ktoré majú za cieľ spríjemniť hru a zlepšiť hráčsky zážitok. V tomto projekte bolo pridaných viacero QoL zmien na zvýšenie plynulosti a odozvy hry.

8.1 Coyote time

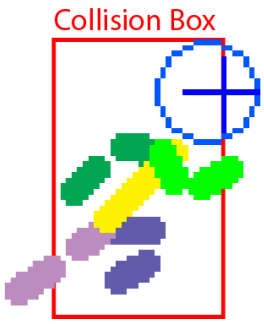
Coyote time je dodatočný čas, počas ktorého môže hráč skočiť, aj keď už fyzicky nie je na zemi. Hoci to môže pôsobiť nelogicky, funguje na konkrétnych podmienkach. Hráč nemôže skočiť iba krátko po opustení zeme. Táto mechanika predchádza frustrácii, keď hra zaznamená vstup oneskorene alebo hráč nepresne odhadne vzdialenosť okraja plošiny o pár pixelov.



Obrázok 13 – Grafické znázornenie funkcie “Coyote time”

8.2 Collision

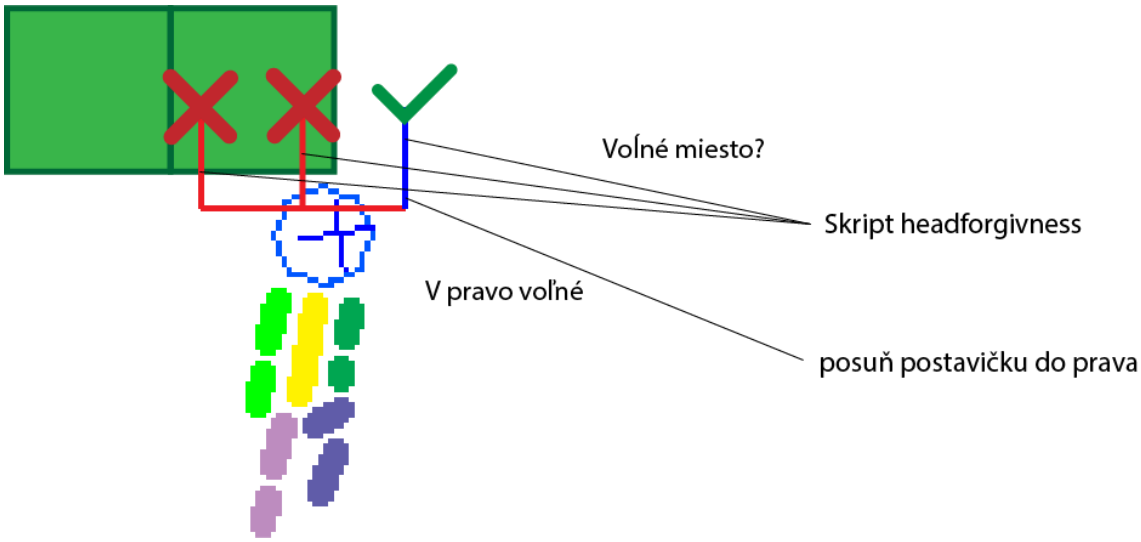
V hrách hráčova vizuálna reprezentácia často nezodpovedá jeho skutočným hraniciam v hernom svete. Na určenie kolízií sa preto využíva hitbox, ktorý presne definuje, kedy a ako sa hráč dotýka okolitého prostredia. V GameMakeri sa na tento účel používa collision box, ktorý slúži na detekciu kontaktu s objektmi v hre a určuje reálne hranice postavy, podľa ktorých systém vyhodnocuje kolízie. To znamená, že hoci hráč na obrazovke vyzerá ako postavička, hra ho vníma ako jednoduchý box. Samo o sebe to nie je problém, no môže to spôsobiť nepresnosti pri pohybe. Napríklad ak hráč vyskočí vedľa steny nad sebou, vizuálne by sa mohlo zdať, že má dostatok priestoru, no ak je collision box širší než postavička, hra môže vyhodnotiť, že sa hráč dotkol steny a zastaví jeho pohyb, aj keď to vizuálne nevyzerá správne.



Obrázok 14 – Grafické znázornenie *Collision Boxu*

Tomuto problému sa môžeme vyhnúť nastavením collision boxu tak, aby presne kopíroval vizuálnu reprezentáciu postavy. Ak by sme však postupovali týmto spôsobom, mohlo by to spôsobiť zasekávanie postavy, najmä pri animáciách, kde sa jej tvar mení – napríklad pri natiahnutí rúk alebo pri skoku. Ak by sa collision box dynamicky menil spolu s animáciou, mohlo by dôjsť k situáciám, kde by sa postava náhle ocitla vo vnútri steny alebo iného objektu. Rovnako by mohla nastať situácia, keď sa pri pohybe postava vizuálne plynule posúva, no ak ruka či iná časť modelu vyčnieva z collision boxu, môže nečakane naraziť na prekážku, čo by pôsobilo nespravodlivo a nepredvídateľne pre hráča.

Problém s buchnutím hlavy vieme vyriešiť bez dodatočného zásahu do základnej kolízie. Skript *headForgiveness* kontroluje, či sa nad hlavou postavičky nachádza voľné miesto – buď doľava, alebo doprava. Ak áno, umožní hráčovi pokračovať v pohybe bez toho, aby ho hra nepríjemne zastavila kvôli malej nepresnosti v kolízii.



Obrázok 15 – Zjednoduchčené grafické znázornenie skriptu *HeadForgrivness*

8.3 Medzery

Kód, ktorý zastavuje hráča, najprv zisťuje, či by sa v ďalšom kroku nachádzal v stene. Ak áno, pohyb sa zastaví. Tento prístup môže spôsobiť nechcenú situáciu, keď napr. má hráč rýchlosť 5, ale medzi ním a stenou zostáva iba medzera veľkosti 3, hrač zastaví ešte pred viditeľné ďaleko stenou a to pôsobí neprirodzene.

Riešenie je jednoduché – ak by hráč skončil v stene, dodatočný skript ho postupne prisunie bližšie k stene, až kým sa jej presne nedotkne. Maximálna vzdialenosť, ktorú pri tom kontrolujeme, je rovná jeho rýchlosti, keďže väčšia vzdialenosť by už v predchádzajúcej detekcii kolízie nebola zachytená.

8.4 Nastavenia

V projekte sa nachádzajú aj nastavenia, ktoré umožňujú hráčovi upraviť hlasitosť hry pre väčší komfort. Sú rozdelené do troch kategórií zvukov a obsahujú aj hlavný ovládač hlasitosti (master volume). V pauzovom menu je tiež dostupné tlačidlo na zmenu kostýmov s možnosťou meniť kostým inputom doprava alebo doľava a tlačidlo s možnosťou reloadovať (načítať) posledný checkpoint, ak by sa hráč ocitol v situácii, kde nemôže pokračovať ďalej.

9 Gameplay

9.1 Scény

Hra síce predstavuje jeden veľký kontinuálny svet, no kvôli optimalizácii a komfortu hráča je rozdelená na jednotlivé časti. Ak sa hráč chce vrátiť do určitej časti sveta bez nutnosti prechádzať predchádzajúce úseky, má možnosť tak urobiť.

9.1.1 Scéna 1

Scéna jedna (Nekonečné klesanie) slúži ako úvodná časť hry, ktorá hráča oboznamuje s ovládaním. V tejto fáze sa hráč naučí pohybovať postavou, skákať, a používať základné akcie, ktoré budú kľúčové počas celej hry. Okrem toho upútava na ďalšie časti hry, čím motivuje hráča pokračovať. Týmto spôsobom je hráč postupne uvedený do herného sveta, čo mu umožňuje hladko začať a získať základnú orientáciu.

9.1.2 Scéna 2

Scéna dva (Nedobrovoľná tyrania) je zameraná na rozvoj hráčových schopností a jeho pochopenie v oblasti pohybu V tejto fáze sa hráč naučí efektívne využívať pokročilejšie pohybové schopnosti,. Okrem hlavnej cesty obsahuje scéna aj vedľajšiu cestu, ktorá ponúka výzvu pre hráčov, ktorí hľadajú extra obtiažnosť. Táto vedľajšia cesta motivuje hráčov, aby si osvojili a zdokonalili svoje pohybové schopnosti, čím sa stáva náročnejšou.

9.1.3 Scéna 3

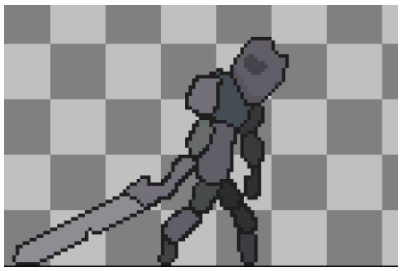
Scéna tri (Šach) sa zameriava predovšetkým na *combat*, ale zároveň prináša aj nové pohybové mechaniky. Táto scéna výzvou preverí všetky zručnosti, ktoré sa hráč naučil doteraz, od pohybovania sa po prostredí až po efektívne využívanie útokov. Ako vyvrcholenie scény hráč čelí „f*inálnemu nepriateľovy*“. Okrem hlavnej cesty scéna obsahuje aj vedľajšiu cestu, ktorá ponúka ťažšie boje a výzvy pre tých hráčov, ktorí hľadajú náročnejší zážitok.

9.2 Nepriatelia

Nepriatelia sú inšpirovaní šachovými figúrkami a herným prostredím.

9.2.1 Pešiak

Pešiak je základný nepriateľ, pomalý, ale má viac životov. Reprezentuje najnižšiu hierarchikú triedu medzi nepriateľmi, čo je viditeľné v jeho nedokonalosti a náhodných kameňoch a tvaroch. Objavuje sa v druhej a tretej scéne.



Obrázok 16 – Postavička Pešiak

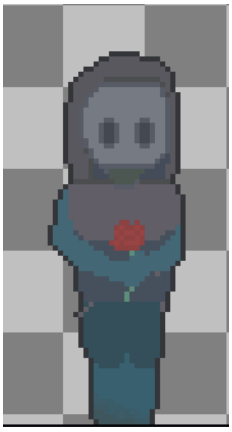
9.2.2 Strelec

Strelec je nepriateľ, ktorý utočí na diaĺku. Má vyššiu hierarchickú hodnotu ako pešiak, ale stále je pod kráľom. Jeho dizajn odráža túto hierarchiu, má krajší výzor než pešiak, ale stále mu chýbajú ruky. Objavuje sa v tretej scéne.

Obrázok 17 – Postavička Strelec

9.2.3 False prophet

Nepriateľ False Prophet je symbolicka mimika hráča. V hernóm príbehu ho vytvorili nešťastní kamenní ľudia, ktorí sa chcú vzbúriť proti kráľovi. Považuje sa za bossového nepriateľa. Objavuje sa v druhej scéne ako tajný boss a v tretej scéne.




Obrázok 18 a Obrázok 19 Postavička False Prophet

9.2.4 Kráľ

Kráľ je hlavný boss hry, inšpirovaný postavou “Minos Prime” z hry Ultrakill. Má najväčšiu hierarchickú moc, čo sa odráža aj v jeho dizajne, vyzerá najlepšie zo všetkých kamenných ľudí. Nachádza sa na konci tretej scény a vo vedľajšej ceste v tretej scéne.


Obrázok 20 a Obrázok 21 Postavička Kráľ a jeho inšpirácia “King Minos Prime”

9.2.5 Shroomite (Hubáčik)

Hubáčik je malá postavička, ktorá nenasleduje hráča agresívne. Namiesto toho sa pohybuje zo strany na stranu. Táto postavička nahrádza klasický prvok v skákačkách, a to sú sawblades, ktoré sa často točia okolo vyvýšených platforiem ako dynamická prekážka.


Obrázok 22 a Obrázok 23 Postavička hubáčik

Zobudeny hubacik a skrytý hubáčik

9.2.6 Boxovacie vrece

Boxovacie vrece je jednoduchý nepriateľ s minimálnou logikou, ktorý slúži čisto na experimentovanie s útokmi.

Obrázok 24 – Boxovacie vrece

10 Nepoužite adície

10.1 Grapling hook

V hre bola pôvodne zahrnutá mechanika uchopovacieho haku. Prototyp bol hádzací nôž s lanom, ktorý sa zasekol do steny a simuloval zložitú fyziku hojdačky, kde sa lano ohýbalo o steny ako fyzické lano by sa ohýbalo o rohy a priťahovalo pohybujúce objekty či nepriateľov. Nakoniec, po odstránení ovládania myškou a vyriešení niekoľkých problémov so zložitými výpočtami, bola táto mechanika v konečnej verzii projektu odstránená.

10.2 Postavičky

Dodatočné postavičky mali nasledovať tému šachových figúrok. Medzi tieto postavy patrila Kráľovná, ktorá mala byť ďalším hlavným bossom, Kôň, rýchla postava, a Veža, ktorá mala byť viac podporný nepriateľ, ktorý poskytuje ochranu pre ostatných nepriateľov. Mimo šachovej témy sme plánovali pridať aj veľkú Hubu, ktorá mala rozprašovací hubový útok, čím by predstavovala neobvyklú hrozbu.

10.3 Prechádzanie časom

Pôvodna hlavná myšlienka hry sa točila okolo časovej manipulácii, hrač dokázal zaznamenávať svoje kroky a vrátiť sa späť v čase o niekoľko sekúnd. Táto mechanika sa však neskôr ukázala ako nezábavná, a preto bola odstránená. Jej téma však pretrvala v hre v podobe motívu VHS, ktorý reprezentuje myšlienku, že hráč nikdy naozaj nezomiera, ale stále neúnavne skúša prekonať prekážky, až kým sa nezlepší.

Záver

Výsledkom maturitnej práce je plne funkčná a hrateľná hra, ktorá splnila všetky stanovené požiadavky. Ponúka imersívny zážitok pre hráčov, ktorí si chcú zahrať krátku, jednoduchú hru, no zároveň poskytuje dostatok obsahu aj pre tých, ktorí sa radi venujú jej detailnému preskúmavaniu a zdokonaľovaniu svojich schopností.

Počas vývoja sme získali množstvo cenných skúseností. V budúcich projektoch by sme sa viac zamerali na stabilitu už existujúcich mechaník pred pridávaním nových, čo by umožnilo lepšie vyváženie hrateľnosti a celkový zážitok hráča. Ukázalo sa, že niekedy je dôležitejšie zdokonaliť to, čo funguje, a ponechať ostatné nápady na budúce iterácie.

Taktiež sme sa naučili efektívnejšie spracovávať spätnú väzbu. Namiesto bezhlavého počúvania každého návrhu je dôležité analyzovať ako daná zmena ovplyvní celkovú víziu projektu. V tomto prípade sme sa od pôvodného konceptu mierne odklonili práve kvôli nesprávne spracovanej spätnej väzbe, čo je ponaučenie do budúcnosti.

Reakcie od testovateľov boli celkovo pozitívne. Hráči, ktorí nemajú skúsenosti s platformovými hrami, ocenili prístupnosť a plynulé učenie sa herných mechaník, zatiaľ čo skúsenejší hráči si užili nepovinné cestičky, ktoré ponúkali extra výzvu.

Získali sme množstvo nových zručností v oblasti grafického dizajnu, level dizajnu aj programovania. Tento projekt nám poskytol cenné skúsenosti, ktoré využijeme pri budúcich tvorivých výzvach.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV / ZDROJE

[1] Final Punch Games – *UV Pixel maping shader* a obrázok vizualizácií UV mapingu Dostupné na: https://final-punch-games.itch.io/pixel-mapping-shader-for-gamemaker

[2] vdDweller 12.11.2016 – Technika color maping a obrazok vizualizácie techniky colo maping Dostupné na: https://forum.gamemaker.io/index.php?threads/sprite-real-time-multi-recoloring-using-shaders.12601/

[3] Fillip Radivojevic 3.11.2023 – Obrázok vizualizácie techniky parallax z článku na stránke RenderHub, Depth in Motion: The Parallax Effect in Games Dostupné na: https://www.renderhub.com/blog/depth-in-motion-the-parallax-effect-in-games

Gizmo199 – Základ VHS efektu použitý v hre, *shader* “Hororify” Dostupné na: https://github.com/Gizmo199/horri-fi

RJ Pasin – Zvukové *sample* použité pre vytvorenie hudby “Main menu hudba” a “Checkpoint tune” Momentálne nedostupné

BDragon1727 – Pixel art efekty použité pri skákani a dopadu Dostupné na: https://bdragon1727.itch.io/

Adobe – Enviromentalné zvuky použité v projekte Dostupné na: https://www.adobe.com/products/audition/offers/adobeauditiondlcsfx.html

Blue Mamoth Games – Gitarové zvuky použité pre útoky, nahraté od postavičky “Munin” v hre Brawlhalla hra Dostupná na: https://www.brawlhalla.com/