Dokumentace – Hair Pandemic

Obsah

[Anotace 2](#_Toc77950775)

[Použité nástroje 2](#_Toc77950776)

[Google Play release 2](#_Toc77950777)

[Pořadí scén 3](#_Toc77950778)

[Fungování scén 4](#_Toc77950779)

[Pohyb jednotek 7](#_Toc77950780)

[Návod na rozšíření 9](#_Toc77950781)

[Nápady na budoucí vývoj 11](#_Toc77950782)

Autor

Lukáš Caha

Pro předmět

NPRG045 – Ročníkový projekt

Matematicko-fyzikální fakulta Univerzity Karlovy, Softwarové a datové inženýrství

Anotace

Tato hra je navržena exkluzivně pro mobilní zařízení a zpracovaná na platformu Android. Jedná se o hru s nižší obtížností vhodnou pro široké spektrum hráčů.

Cílem tohoto ročníkového projektu bylo vytvořit hru s ovládáním a hratelností přizpůsobenými mobilní platformě. Herní mechaniky této aplikace jsou zaměřené na práci se skupinou herních postav, které hráč ovládá joystickem. Pohyb skupiny autenticky napodobuje stádové chování. Proces hledání vhodného algoritmu je popsán zde v dokumentaci.

Projekt je i jednoduše rozšiřitelný o nové jednotky a úrovně a postup pro rozšíření je popsán v závěrečné části této dokumentace.

Použité nástroje

K naprogramování byl použit jazyk C# verze .NET Standard 2.0

Hra byla vytvořena pomocí herního enginu Unity 2020.1.6f1

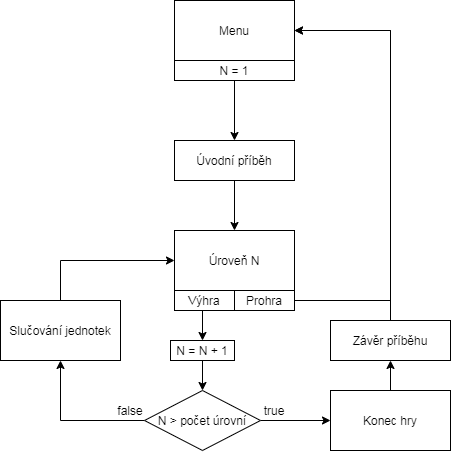
Pro grafiku byl převážně použit GIMP2 a MS paint

Google Play release

Hra je vydaná na Google Play. Pro převedení Unity projektu na soubor .aab (Android app bundle) je potřeba:

* Zakliknutí „Build app bundle“ v „build settings…“ menu
* Vyplnění „Publishing setting“
  + Pojmenování balíčku
  + Určení verze – pro každou novou verzi je nutné inkrementovat manuálně, jinak to GP odmítne
  + Nastavení dostupných rozlišení a rotací
  + Přidání náhledového obrázku
  + Vytvoření Keystoru na podepsání balíčku
* Po buildu stačí nahrát balíček do Dev console, vyplnit produktovou stránku (popisky a fotky ze hry) a udělat dotazník na věkové doporučení hry, poté stačí vyčkat několik hodin až dní a hra je veřejně dostupná na GP

Pořadí scén

Hra dodržuje určité pořadí scén, které zaručuje plynulý průběh. Předpokládaný průběh je **Menu > Úvodní příběh > Všechny úrovně > Závěr příběhu > Menu**.

Chod tohoto diagramu zaručují následující skripty.

Change\_scene.cs

Skript určený na přichycení na tlačítka. Umožňuje zpracovat událost, která má pouze změnit scénu. Má následující použití:

* Tlačítko „Play“ v menu
* Tlačítko „Return ve všech herních úrovních“
* Tlačítko „Continue“ v příběhových scénách

Marge\_dragNdrop.cs

V tomto sktiptu pro účely přesunu mezi scénami je pouze metoda **void EndMerging()**. Tato metoda uloží konečný stav skupiny jednotek a načte další úroveň.

PlayerWon.cs

Pokud počet živých nepřátelských skupin dosáhne nula, tento skript posune hráče do scény se slučováním jednotek.

PlayerLost.cs

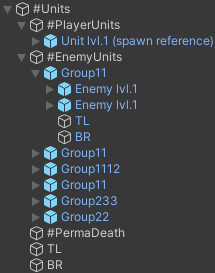
Pokud počet živých hráčských jednotek dosáhne nula, tento skript načte scénu s oznámením prohry.

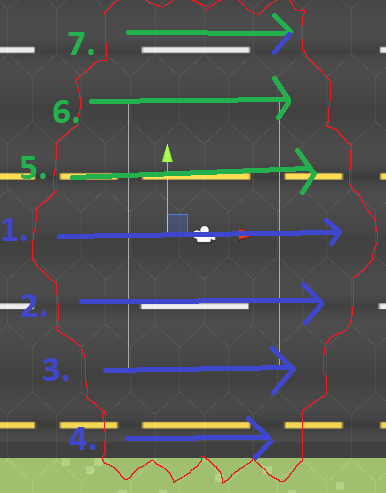
Fungování scén

V následujících kapitolách je popis toho, jak fungují jednotlivé herní mechaniky v určitém typu scén.

Scény úrovní

Úrovně jsou jediné místo kde se děje nediskrétní simulace typická pro hry. Scénu můžeme rozdělit do tří kategorií, kde každá obsahuje určité objekty, jejichž chování je popsáno skripty:

* Prostředí
  + Pozadí
    - Statický 2D obrázek bez nutnosti použití skriptů
  + Překážky
    - Převážně domy – jejich collider nedovolí jednotkám vstoupit
  + Hranice
    - Statické bez nutnosti skriptů
    - Dálnice – každé auto vyrazí s časovým odstupem a po parametrizované vzdálenosti se vrátí na startovní pozici
    - Řeka – obsahuje dva identické segmenty, které se pohybují ve stejném směru a cyklicky se vrací na pozici o dvě délky zpět
  + Pasti – nebezpečné pro obě frakce
    - Hrábě – šlápnutí na ležící hrábě dává poškození rovné 80 % z maximálních životů jednotky, když na nich nikdo nestojí, spadnou na zem
    - Past na medvědy – první jednotka, která stoupne do pasti zůstane uvězněná a přijde o 50 % z maximálního množství životů
    - Autobus – když do jednotky narazí autobus, dostane poškození rovné 90 % z maximálního počtu životů, autobus zatáčí na značkách ve scéně
* Jednotky
  + Hráčské – ovládané třemi skripty
    - UnitBOIDS.cs – pohyb jednotek
    - Stats.cs – udržuje stav životů a způsobuje útok na jednoho nepřítele v dosahu v pravidělných časových intervalech
    - Fraction.cs – pokud jednotka dosáhne 0 životů, tento skript jí zamění za holou neutrální jednotku
  + Nepřátelé
    - Pohyb
      * Nadřazený objekt skupiny obsahuje skript Order\_machine.cs, který určuje cíl skupiny a uznává dosažení
      * Jednotky následují směr určený skupinou
    - Stats.cs + Fraction.cs stejné, jen smrt mění jednotku na hráčskou
  + Holé jednotky – při ostříhání hráčské jednotky vznikne neutrální nepohyblivá jednotka s holou hlavou
* Herní manažeři (game managers) – sedí na libovolném objektu ve scéně a řídí svůj úkol
  + HardBoundaries.cs – udržuje seznam všech jednotek a vrací je zpět do hranic úrovně
  + Player\_spawner.cs – při začátku úrovně vyrobí správné množství hráčských jednotek na určené místo v úrovni

Scéna se slučováním jednotek

V této scéně si hráč vyrobí silnější jednotky a zároveň, jelikož se data o jednotkách přenáší pouze jako počty, tak jsou nové jednotky v další úrovni obnovené na plné zdraví.

1. Načte jednotky, které musí zobrazit v šestiúhelníkové mřížce a vypočítá její velikost
2. Začne plnit mřížku dolní polovinou a následně doplní i horní
3. Při tažení jednotky může hráč pustit tah na jiné jednotce stejné úrovně a tím dojde ke sloučení do vyšší úrovně, jinak se puštěná jednotka vrátí na své místo
4. Při sloučení je vidět zelený efekt spreje
5. Při kliknutí na tlačítko pokračovat se uloží aktuální stav jednotek a zapne se další scéna

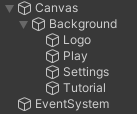
Přenos dat o jednotkách by nebyl možný bez skriptu PlayerWon.cs, který na konci úrovně před načtením merge scény uloží stav v podobě řetězce do PlayerPrefs. Zvolil jsem tuto variantu, protože byla jednoduší než ukládat data do DontDestroyOnLoad objektu. Kdybych potřeboval přenášet jiná data, než čisté počty, bylo by poté rozumnější uložit je do samostatného objektu.

Z merge scény se data o jednotkách přenáší stejným způsobem a na začátku další úrovně se tím řídí skript Player\_spawner.cs, který vytvoří patřičné nové jednotky.

Scény s příběhem

Obě scény s příběhem mají přichystané sekvenčně všechny snímky za sebou. Jediná interakce, co je možná je kliknutí, které odstraní vrchní obrázek, čímž se hráč dostane na poslední snímek, který odkazuje na další scénu. Příběh lze tedy za pouhého přidání snímků či zkopírování scény rozšířit. Jelikož je každý obrázek gameObject, není mimo ani přidání textu, nebo dokonce animace jako podobjekt.

Menu

Menu je jednoduchá rozbočovací obrazovka obohacená o mini-animace nad několika elementy uživatelského rozhraní. Umožňuje pouze tři akce, z nichž je jedna tajná.

První akcí je tlačítko Play, které zapne úvodní příběh, který poté zapne první úroveň.

Druhou akcí je zobrazení nápovědy. Jedná se o příběhovou scénu se snímky popisující, jak slučovat jednotky, což je nejsložitější mechanika ve hře.

Třetí tajná akce je zrychlení hry pro účely ladění. Tlačítko zodpovědné za tuto akci je schované pod písmenkem „H“ v nadpisu. Maximální zrychlení je 8x.

Přechody mezi scénami

Většinou když se změní scéna, musí dojít k nějakému přenosu dat. Už jsme si ukázali v kapitole o slučování jednotek, jak se přenáší počty hráčských jednotek. Další, trochu složitější přenos dochází u objektu #UnitAuthority se skriptem Unit\_list.cs.

Skript Unit\_list.cs obsahuje ve Start() metodě příkaz DontDestroyOnLoad(this), který zamezí tomu, aby se objekt na kterém je skript připnutý ztratil při přechodu mezi scénami. Takto stačí umístit jediný objekt #UnitAuthority do první úrovně a následně ho lze používat ve všech scénách, které se načtou poté.

Objekt je navržený pomocí návrhových vzorů singleton a mediator. Jeho hlavní funkce je přenášet informace o jednotkách, aby každá jednotka při změně frakce nemusela mít uloženou svou následující formu u sebe, ale stačí se zeptat mediátora. Objekt je také singleton, protože je nutné mít pouze jeden takovýto objekt.

Poslední poznámka se týká křížku v názvu objektu. Označení „#JmenoObjektu“ je mé znamení pro objekty na jejichž jméně záleží, protože někde v kódu se používá funkce GameObject.Find(string), která vrátí objekt s daným jménem. Volání této funkce je poměrně drahé, protože roste lineárně s počtem objektů ve scéně, a proto je lepší (pokud to lze) předávat reference na tyto speciální objekty jinak. Téměř vždy je volaná ze Start metody, která se volá pouze jednou (🡪 složitost O(n2)), tudíž nemůže nastat výrazný pokles ve výkonu oproti detekování fyzikálních kolizí (mnoha jednotek), které také běží v čase O(n2).

Pohyb jednotek

Během vývoje jsem vyzkoušel několik algoritmů pro pohyb hráčských jednotek. Zde porovnám jejich výhody a nevýhody.

Leader driven groups

První způsob chování jsem postavil na základě rozdělení jednotek do skupin a každé jsem přiřadil silnějšího vůdce, ke kterému všechny jednotky ze skupiny šly. To drželo skupinu soudržnou v kruhovém útvaru a umožňovalo abstrahovat pohyb skupiny do pohybu jedné jednotky.

* Výhody
  + Nejjednodušší algoritmus udržující jednotky v soudržné skupině
  + Při zvolení správného způsobu pohybu objektů a augumentace fyzikální váhy objektů lze řídit, které jednotky budou ve středu a které na okraji skupiny 🡪 možnost více ovládat strategii pomocí uspořádání jednotek
* Nevýhody
  + Nutnost dvojího kódu pro jednotky a vůdce
  + Po smrti vůdce neměly jednotky kam jít
    - Vzhledem ke zvýšené síle nemůže skupina zvolit nového vůdce
    - Kdyby byly všechny jednotky stejně silné, může probíhat předávání vůdcovské role, ale každá změna vyžaduje aktualizování reference všem jednotkám

Závěr

Tuto strategii jsem vyzkoušel na začátku vývoje a kvůli nevýhodám jsem jí prioritně změnil na následující strategii pohybu. Když bych měl pojednat o věrohodnosti davového chování, tak bych rozhodně vytkl nerealističnost následování vůdce.

Move in direction + regroup

Tento algoritmus je spíše oprava předchozí strategie, než pokus o nejlepší řešení. Tento algoritmus sestává z dvou fází: (i) pohyb ve směru joysticku; (ii) seskupení k průměrnému bodu, pokud není joystick ovládán.

* Výhody
  + Každý jedinec ve skupině funguje nezávisle na ostatních
  + V teorii obsahuje tato strategie vše potřebné pro kontrolu nad davem
* Nevýhody
  + Při pohybu dochází k mírnému štěpení skupiny, tudíž je nutné, aby se hráč zastavoval a nechal jednotky seskupit
  + Všechny jednotky se snaží dostat k průměrnému bodu, což způsobí tlačení, což se odchyluje od reálného chování davu při seskupování

Závěr

Tato strategie zůstala ve hře po celou dobu vývoje, protože byla adekvátní pro plnou hratelnost. Nahrazení proběhlo až těsně na konci, kdy jsem vyžadoval nadstandartní chování. Algoritmy fungují jinak, než uvažuje reálný dav, ale jde o dobrou aproximaci chování.

BOIDS

Jde o algoritmus Craiga Reynoldse: <https://en.wikipedia.org/wiki/Boids>

Tento poslední algoritmus nese název od slova bird, konkrétně bird-oid object. Simuluje stádové chování, kde je rozhodování distribuované mezi jedince, kteří svou individuální reakcí na podněty tvoří soudržné stádo. Chování jednice můžeme rozdělit na tři složky:

1. Soudržnost (cohesion)
2. Držení vzdálenosti (separation)
3. Srovnání směru (alignement)

Každá složka udává vektor, kudy chce jedince poslat a jejich lineární kombinace tvoří výsledný směr pohybu. Aby šlo skupinu ovládat a učinil jsem jí ještě o trochu důvěryhodnější, přidal jsem další dvě složky: (i) směr joysticku a (ii) náhodný vektor.

Kromě koeficientu pro součet vektorů jsou zavedené také koeficienty dosahu, což simuluje že se boid nemůže řídit podle jiného boida, kterého ani nevidí.

* Soudržnost
  + Zaznamená ostatní boidy v okruhu 2\*R
  + Vypočítá aritmetický průměr jejich pozic
  + Vrátí vektor směřující k průměrné pozici
* Držení vzdálenosti
  + Zaznamená ostatní boidy v okruhu R
  + Vypočítá směr od druhého boidu a vydělí ho čtvercem vzdálenosti
  + Vrátí součet těchto směrů od ostatních boidů
* Srovnání směru
  + Zaznamená ostatní boidy v okruhu R
  + Vyčte z nich jejich směr pohybu
  + Vrátí aritmetický průměr těchto směrů
* Vstup joysticku
  + Vrací vektor, ve kterém hráč namířil joystickem
* Náhodný vektor
  + Vrací plně náhodný vektor
  + Odstraňuje determinizmus u jednotek, které jsou na stejné pozici

Použité koeficienty

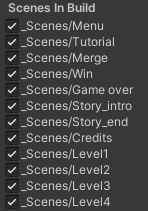
* Rychlost = 15
* R = 500 (na obrázku 100, záleží na velikosti úrovně)
* Skupinový boids koeficient = 0,1
* Koeficient separace = 0.5
* Koeficient koheze = 0.005
* Koeficient srovnává = 0.005
* Koeficient joysticku = 0.5
* Koeficient náhodnosti = 0.005

Závěr

Fungování na základě distribuované inteligence a jednoduchých algoritmů, podle kterých se jedinci řídí nejlépe napodobuje skupinové chování, a proto je tento algoritmus finální.

Návod na rozšíření

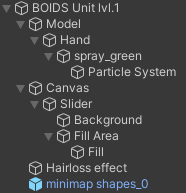
Už od začátku je hra navržena tak, aby šla velice jednoduše rozšířit o nové jednotky a úrovně. Zde je popsán způsob, jak toho dosáhnout.

Přidání úrovně

Nejjednodušší způsob je zkopírovat poslední existující úroveň, která obsahuje veškeré potřebné věci na chod hry a změnit jméno z „Level4“ na „Level5“. Následně stačí pouze přidat novou scénu do *File > Build settings... > Scenes in Build*. Následné úpravy pak lze provést na této kopii. Je však nutné zachovat tyto objekty:

* Camera rig – míří hlavní kameru na všechny hráčské jednotky
* Minimap camera – zabírá pohled na celou úroveň, při zvětšení je nutné zvětšit dostatečně velikost podobjektu Camera
* PlayerSpawner – drží skript, který vyrobí hráčské jednotky na začátku úrovně, samotné jednotky se ale naspawnují na lokaci kde je jediná jednotka přítomná jako podobjekt objektu *#Units > #PlayerUnits*, jednotka určující lokaci bude smazána, pokud bude přítomno více jednotek, dostane je hráč jako bonus počínaje touto úrovní
* Controls Canvas – obsahuje veškeré UI potřebné pro ovládání hry
  + Minimap – zobrazení minimapy
  + #Joystick – hráč tímto ovládá své jednotky
  + Pause button – tlačítko na pozastavení hry a otevření panelu s možnostmi
  + Paused panel – panel obsahující tlačítka zpět do menu a pokračovat ve hře
* #Units – obsahuje veškeré jednotky ve hře a nějaké chování spojené s nimi
  + TL a BR – markery značící z jakého území nesmí jednotky vyskočit, nelze jiný tvar než obdélník, ale to už lze upravit tradičními Unity collidery
    - TL = top left
    - BR = bottom right
  + #PlayerUnits – obsahuje hráčské jednotky, pokud dosáhne počet objektů 0, hráč prohraje, nelze tedy přidávat jednotky, které budou mít nastavené isActive = false
  + #EnemyUnits – obsahuje skupiny nepřátelských jednotek, pokud počet skupin dosáhne 0 vyhrává hráč, každá skupina obsahuje:
    - Jednotky – pokud dosáhne počet jednotek 0, skupina se vymaže
    - TL a BR – těmito markery lze vymezit menší území, než je hrací plán a tím určit teritorium dané skupiny
    - Skupina se dá nastavit pomocí parametrů WaitTime a ArrivalPrecision
  + #PermaDeath – trvale mrtvé jednotky, slouží pouze jako úschovna
* EventSystem – umožňuje konání input eventů pro UI elementy

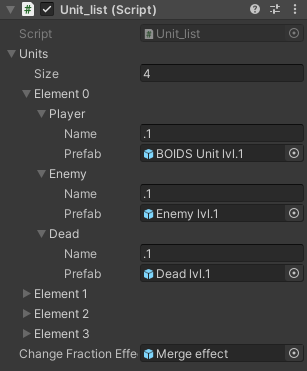
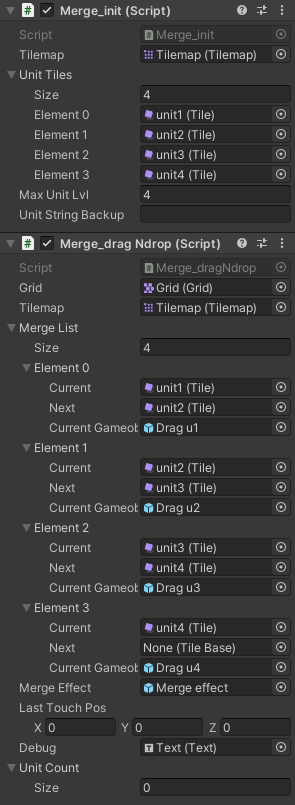
Zbytek úrovně je pouze pro vzhled, a tudíž není nutné dodržovat žádnou strukturu. Je možné přidat pasti a collidery. Ty zajistí že jednotky nebudou chodit například přes budovy.

Přidání jednotky

Pokud chceme přidat jednotku, musíme vytvořit minimálně 2 varianty. V ideálním případě přidání by byla vytvořena nepřátelská jednotka, hráčská jednotka a ostříhaná jednotka všechny jedné úrovně. Lze ale vynechat nepřátelskou variantu, a tak vznikne jednotka, kterou lze získat mergem, ale nelze získat konverzí.

Následují kroky, které je nutné následovat, pokud chcete přidat novou jednotku:

Hráčská jednotka

1. Vytvořte kopii jedné z jednotek ve složce *Unit Prefabs > Player > BOIDS units*
2. Zaveďte veškeré změny
   1. Model
      1. Změnit texturu
      2. Změnit texturu ruky
      3. Změnit efekt spreje/jiného zdroje barvy
   2. Statistiky
      1. Maximální životy
      2. Útočná síla
      3. Interval útoku
   3. Pohyb
      1. Rychlost
      2. Hranice, kam jednotka může
      3. Nedoporučuji upravovat žádné koeficienty v BOIDS skriptu
3. Prefab nové jednotky zaveďte do *#UnitAuthority* objektu ve scéně *Level1*
4. Ve scéně Merge je nutné zavést ještě dvě úpravy
5. Ve složce *Assets > Tiles* je nutné vytvořit kopii libovolné jednotky a nahradit sprite vzhledem nové jednotky
6. Tento asset zaveďte to skriptu *Merge\_init* na objektu *Merge engine* ve scéně Merge a také do skriptu *Merge\_dragNdrop* na stejném objektu
7. Posledním krokem je vytvoření assetu *Drag u5* kopírováním z *Drag u4* a nahrazením spritu na vzhled nové jednotky, následně tento asset přiřaďte na správné místo ve skriptu *Merge\_dragNdrop*

Oholená jednotka

1. Vytvořte kopii jedné z jednotek ve složce *Unit Prefabs > Dead*
2. Změňte sprite na požadovaný vzhled
3. Nahraďte stopu pláče specifickou pro tuto jednotku (nepovinné)
4. Zaveďte jednotku do objektu *#UnitAuthority* objektu ve scéně *Level1*

Nepřátelská jednotka

1. Vytvořte kopii jedné z jednotek ve složce *Unit Prefabs > Enemy*
2. Opakujte kroky 2 a 3 ze zavedení nové hráčské jednotky

Nápady na budoucí vývoj

Kromě možnosti jednoduše přidávat další úrovně a jednotky je zde ještě několik mechanik, které nejsou potřeba pro plně funkční hru, avšak by bylo zajímavé jejich přidání.

* Zacyklení příběhu přes další barvy vlasů:
  + Náznak je již přítomný v posledním snímku příběhu na konci hry
  + Po odehrání zelené barvy by mohla následovat modrá a jiné neobvyklé barvy, rozhodně bych nevyloučil ani hnědou, se kterou všechny postavy začínají.
  + Bylo by možné zavedení rouge like mechanik. Po dohrání hry by se mohla na další běh zvednout obtížnost a hráč by získal bonusy, které by mu pomohly projít snadněji.
* Procedurálně generované úrovně
  + Assety použité na stavbu úrovně vesnice a města jsou dlaždicové, tudíž by se úrovně mohly generovat automaticky a vždy jinak.
* Možnost přeskládat jednotky
  + První dvě strategie pohybu umožňovaly udržovat skupinu v určitém uspořádání
  + Toho by se dalo využít a umožnit hráči v merge fázi poskládat jednotky do vrstev
  + Navíc by pomocí úpravy rychlosti jednotek šlo určit, které jednotky se budou držet vepředu pohybujícího se davu
* Aktivně útočící jednotky
  + Aktuálně jednotka útočí pouze pokud je v dosahu nepřítele
  + Útočení nemá žádný vliv na pohyb
  + Pro konzistentnější útoky by šla přidat „přitažlivost“ směrem k jednotce na kterou aktuálně útočí
  + Takovou sílu by bylo samozřejmě možné převážit, aby šly dvě bojující jednotky odtrhnout