STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
PREŠOV

IV.SA - 02

ŠK. R. 2018 – 2019

PČOZ – HRA PRE ANDROID - JUMPAROUND

PAVOL BARAN

Konzultant: Ing. Mária Hedvigová

Anotácia v slovenskom jazyku

Cieľom tohto projektu bolo vytvoriť hru pre viacerých hráčov na platformu Android. Zariadenia medzi sebou budú komunikovať prostredníctvom lokálnej siete. Vďaka tomu môže každý užívateľ hrať našu hru na vlastnom mobilnom zariadení. Zistíme aké praktiky sa využívajú pri synchronizovaní hier pre každého užívateľa a rozoberieme si spôsob synchronizovania v našej hre. Vysvetlíme si rozdiely medzi TCP a UDP protokolmi, a prečo sme sa rozhodli pracovať práve s UDP a ENet knižnicou. Keďže sme si pre tvorbu našej hry vybrali herný engine Godot, povieme si aj o práci s jeho súčasťami. Na záver si rozoberieme časti našej hry a vzhľad užívateľského rozhrania.

Anotácia v anglickom jazyku

The goal of this project was to create a multiplayer game for the Android platform. Devices will communicate with each other through the local network. This allows every user to play our game on their own mobile device. We'll find out what practices are being used to synchronize games for each user and discuss how we sync our game. We explain the differences between TCP and UDP protocols, and why we decided to work with UDP and ENet library. Since we chose Godot game engine to create our game, we will also talk about working with its components. Finally, we will discuss parts of our game and the look of the user interface.

Čestné vyhlásenie

Vyhlasujem, že celú prácu stredoškolskej odbornej činnosti na tému Hra pre android - JumpAround som vypracoval/a samostatne, s použitím uvedenej literatúry.

Som si vedomý zákonných dôsledkov, ak v nej uvedené údaje nie sú pravdivé.

Prešov, 24. apríl 2019 .....

vlastnoručný podpis

Poďakovanie

Týmto by som sa chcel poďakovať mojej konzultantke Ing. Márii Hedvigovej za jej odbornú pomoc, know – how a trpezlivosť pri realizácii tohto projektu.

Obsah

Úvod

Už odmalička ma fascinovali počítačové hry. Svoj voľný čas som častokrát venoval sedeniu za počítačom a objavovaniu nových virtuálnych svetov. Spočiatku to boli malé „flešovky“ a podobné hry nájdené na internete. Neskôr som ale objavil hry, ktoré vyvíjali a publikovali multimiliónové spoločnosti. Uvedomil som si aká rozsiahla je herná scéna.

V dnešnej dobe, v ktorej sa z profesionálnych hráčov stávajú celebrity sa otvárajú množstvá pracovných príležitostí pre ľudí, ktorý majú záujem o tento odbor. Od programátora po beta testera, existuje množstvo slušne zarábajúcich pozícií.

Ja osobne som chcel vedieť prečo a ako hry fungujú. To, spoločne s ďalšími dôvodmi je prečo som si vybral túto prácu. Človek sa najlepšie naučí praxou a preto som chcel vytvoriť vlastnú počítačovú hru.

Svoju hru som chcel hrať spoločne s priateľmi. Keďže sedenie viac ako dvoch ľudí za počítačom je nepríjemné, plánoval som vytvoriť hru, ktorá by sa dala hrať prostredníctvom siete, bola by časovo nenáročná a kompetetívna.

Očividným riešením je objektové programovanie. Na realizáciu svojho projektu som si preto vybral MIT licencovaný herný engine Godot. Vďaka nemu som nemusel „znovu objavovať koleso“ a mohol som sa sústrediť na vymýšľanie hernej logiky namiesto vymýšľania low – level architektúry .

Pretože som doteraz nikdy nerobil herný projekt takýchto rozmerov, musel som sa zoznámiť s pojmami ako sú kolízie, RPC, klient/server a mnohými ďalšími, o ktorých si povieme v nasledujúcich kapitolách.

1 Cieľ a metodika práce

Našim cieľom je vytvoriť hru pre viacerých hráčov, v ktorej by komunikácia prebiehala prostredníctvom lokálnej siete.

Na začiatok sme vytvorili iba základný koncept hry. Cieľom hry je zostať nažive ako jediný. Hráči budú vyradený z hry nasledujúco: 1. Dotknú sa zeme, stropu alebo míny, čím prídu o jeden bod zdravia. Ak prídu o všetky tri, budú vyradení. 2. Iný hráč na nich dopadne zhora čím budú okamžite eliminovaný z hry.

Počas hry im v manévrovaní pomôže množstvo elementov. Hráči budú taktiež zbierať orby, vďaka ktorým si budú môcť zväčšiť počet životov a zároveň tieto orby slúžia ako body na určenie celkového víťaza hry.

Ďalším krokom bolo vytvoriť lobby, do ktorého by sa mohli užívatelia pripojiť. V lobby si môže užívateľ vybrať farbu svojej hernej postavičky. Táto farba sa nemôže zhodovať so žiadnou s farbou ostatných užívateľov. Taktiež si bude vedieť zmeniť svoje meno, aby ho ostatní hráči vedeli rozoznať.

Keďže sme chceli aby si užívateľ vedel vybral medzi tým či vytvorí hru, alebo sa do nej pripojí, ďalším logickým krokom bolo implementovať herné menu. Pomocou štyroch tlačidiel sa tak užívateľ mohol pripojiť do hry, vytvoriť hru, prečítať si pravidlá alebo vypnúť hru. Pripojiť sa musí zadaním IP adresy do popup okna. Po nesprávnom zadaní bude upozornený na nesprávny formát adresy alebo na neúspešnosť pripojenia.

Keď už sme mali implementované herné menu, lobby a hru samotnú bolo na čase vytvoriť rozhranie, pomocou ktorého by sa hráči dozvedeli o svojich herných výsledkoch. Po troch herných kolách sa tak rozhranie zmení a ukáže výsledky hry. Zobrazený budú prvý traja hráči: mená farba postavičky, počet výhier a počet orbov. Každý hráč ma možnosť vrátiť sa spať do lobby, alebo vrátiť sa spať do hlavného menu. Po tomto kroku bola naša hra teoreticky hotová. Grafiku do hry sme kreslili počas každej implementácie novej funkcionality. Následne sme hru testovali opravovali chyby a hrali sa s premennými, aby mal hráč čo najlepší zážitok.

2 Úvod do problematiky - Networking

2.1 TCP vs. UDP

V našom projekte budeme pracovať hlavne s protokolom UDP. Je však dobré povedať si o rozdieloch medzi TCP a UDP, ako fungujú a v akej situácii by sme si mali vybrať jeden alebo druhý.

TCP - transmission control protocol (protokol riadenia prenosu) . Je spoľahlivý a „connection oriented“. To znamená, že počítače si najprv musia vytvoriť spojenie pomocou „three-way handshaku“ a až potom si vymieňajú dáta. TCP spojenia sú spoľahlivé a dáta prichádzajú zoradené. Všetky údaje, ktoré odošleme, zaručene dorazia na druhú stranu a v poradí, v akom sme ich poslali. Je to tiež prúdový protokol, takže TCP automaticky rozdeľuje dáta do paketov a posiela ich cez sieť.

UDP - user datagram protocol (používateľský datagramový protokol). UDP tak ako TCP je protokol na prijímanie a odosielanie dát. Od TCP sa ale líši v tom, že je „connectionless“, to znamená že nevytvára spojenie medzi počítačmi a nezaručuje doručenie dát. Takže ak počítač odošle dáta, nezaujíma sa ďalej o to, či dáta do cieľa dorazili alebo nie. V praxi však väčšina paketov, ktoré sú odoslané dorazia. Môže sa ale vyskytnúť 1 - 5% strata paketov a občas sa môže objaviť perióda, v ktorej žiadne pakety nedorazia. Neexistuje taktiež žiadny pojem o poradí paketov. Ak by sme poslali 3 pakety v poradí 1,2,3 mohli by doraziť úplne nezoradene napríklad 3,2,1. V praxi ale majú pakety tendenciu prísť v poriadku väčšinu času. Nemôžeme sa na to ale spoľahnúť. Všeobecne platí, že TCP možno považovať za spoľahlivý a usporiadaný ale pomalý. UDP ako nespoľahlivý a neusporiadaný ale rýchly.

2.2 UDP pri tvorbe hier

Vo väčšine multiplayerových hier sú dáta časovo kritické. To znamená, že hráčovi nezáleží na tom, čo sa stalo pred sekundou, ale čo sa deje práve teraz. TCP na to nie je prispôsobené. Chceme aby sa dáta dostali od klienta na server čo najrýchlejšie a aby sme nečakali zbytočne na preposlanie stratených dáta pretože novšie už sú pravdepodobne na ceste. Preto sa TCP v hrách využíva zriedkavo.

Počas hrania nám teda v niektorých situáciách záleží len na najnovších dátach. To platí napríklad pre stav hráča alebo vstup z klávesnice. Čo mám z toho ak som stlačil pred sekundou tlačidlo a vyčaroval som blesk až teraz? Potom sú tu ale dáta ktoré chceme aby určite dorazili. Tými môžu byť napríklad príkazy na načítanie sveta alebo odobranie života pri skoku z veľkej výšky. Čo teda robiť ak chceme rýchle doručenie dát, ale zároveň kontrolu integrity? Riešením určite nie je používať aj TCP aj UDP, keďže TCP má tendenciu tvoriť stratu UDP paketov. Najlepším riešením je implementovať niektoré vlastnosti TCP na upravenom UDP založenom protokole. O to presne sa stará knižnica ENet.

2.3 ENet knižnica

Godot používa mid-level objekt NetworkedMultiplayerPeer. Tento objekt nie je určený na prime vytvorenie, ale je navrhnutý tak, aby rôzny ho počet implementácií mohlo poskytnúť.

Tento objekt rozširuje PacketPeer, takže zdedí všetky užitočné metódy na serializáciu, odosielanie a prijímanie dát. Okrem toho pridáva metódy na nastavenie peerov, prenosný režim, atď. Tiež zahŕňa signály, ktoré nám umožní vedieť, kedy sa odpojil alebo pripojil peer. V predvolenom nastavení poskytuje Godot implementáciu založenú na ENet (NetworkedMultiplayerEnet).

ENet poskytuje relatívne tenkú, jednoduchú a robustnú sieťovú komunikačnú vrstvu na vrchole UDP (User Datagram Protocol). Primárnou vlastnosťou, ktorú poskytuje, je voliteľné spoľahlivé doručovanie paketov v poradí.

ENet vynecháva určité funkcie na vyššej úrovni siete, ako napríklad autentifikáciu, objavovanie serverov, šifrovanie alebo iné podobné úlohy, ktoré sú špecifické pre rôzne druhy aplikácií. Knižnica tak zostáva flexibilná, prenosná a ľahko implementovateľná.

Class: NetworkedMultiplayerPeer
Inherits: PacketPeer , Reference , Object
Inherited by: NetworkedMultiplayerENet

Public Methods:

```
    int   get_connection_status() const
    int   get_packet_peer() const
    int   get_unique_id() const
   bool   is_refusing_new_connections() const
   void   poll()
   void   set_refuse_new_connections( bool enable )
   void   set_target_peer( int id )
   void   set_transfer_mode( int mode )
```

Signals:

```
connection_failed()
connection_succeeded()
peer_connected( int id )
peer_disconnected( int id )
server_disconnected()
```

Constants:

```
TRANSFER_MODE_UNRELIABLE = 0
TRANSFER_MODE_UNRELIABLE_ORDERED = 1
TRANSFER_MODE_RELIABLE = 2
CONNECTION_DISCONNECTED = 0
CONNECTION_CONNECTING = 1
CONNECTION_CONNECTED = 2
TARGET_PEER_BROADCAST = 0
TARGET_PEER_SERVER = 1
```

Obr. 1 Trieda NetworkMultiplayerPeer (Zdroj: https://docs.godotengine.org/en/3.1/tutorials/networking/high_level_multiplayer.html, 2019)

3 Sieťové modely a synchronizácia

Na to aby užívatelia videli na obrazovkách to isté sa používali rôzne metódy. Je dôležité aby sme si vybrali tú správnu pre nás. V tejto časti si povieme o modeloch, ktoré boli používané, a ktoré sú používané dodnes.

3.1 Peer to peer lockstep

Na začiatku boli hry synchronizované pomocou peer-to-peer pripojenia. Každý počítač si s každým navzájom vymieňal informácie o hre cez kompletne prepojenú mesh topológiu. Základnou myšlienkou bolo premeniť hru na sériu kôl a príkazov, ktoré by po spracovaní ovplyvňovali stav hry. Tieto príkazy mali tvar: pohyb, buduj, znič, útoč. Na to, aby hráči na svojich obrazovkách videli to isté, stačilo vykonať rovnaké príkazy na každom počítači po ukončení kola.

Tento model mal však veľa nedostatkov. Na to aby sa hra odohrala rovnako bolo potrebné, aby boli príkazy jednotlivých hráčov pre dané kolo prijaté na každom počítači pred tým ako mohla hra pokračovať. To znamenalo, že všetci hráči museli čakať na toho kto mal najväčšie oneskorenie. Ďalším nedostatkom bolo to, že na správne zobrazenie na všetkých počítačoch museli hry začínať s rovnakého základného stavu. To robilo pripojenie do už bežiacej hry takmer nemožné. Pre tento dôvod museli byť hráči, ktorí chceli spolu hrať v jednom lobby. Presne takéto lobby máme implementované aj v našej hre, keďže funguje na podobnom princípe.

3.2 Klient/Server

Peer to peer lockstep bol dlho zaužívaný model. To sa zmenilo v roku 1996 keď John Carmack a jeho tím vydali hru Quake, založenú na modeli Klient/Server. Namiesto toho aby každý hráč spúšťal rovnaký kód sa z hráčov stali klienti a komunikovali exkluzívne iba z jedným počítačom zvaným server. Klienti už nemuseli spúšťať rovnakú sadu príkazov pretože hra v skutočnosti existovala iba na servery. Každý klient tak v podstate predstavoval terminál zobrazujúci súčasný stav hry a odosielal vstup z klávesnice. Server tieto vstupy spracovával a odosielal klientom ich súčasný stav v hre.

A presne takto sa správal čistý klient/server model. Klient odosielal serveru dáta zo vstupných zariadení: myš, klávesnica, joystick a server upravil pozíciu/stav hráča v hre a odoslal klientovi informácie o jeho polohe. Klient následne interpoloval svoju minulú pozíciu s novo prijatou pozíciou a pohyb v hre vyzeral prirodzene. Toto bol obrovský skok v hernom priemysle. Teraz sa herný zážitok odvíjal od pripojenia medzi klientom a serverom a nie od peera, ktorý mal v sieti najväčšie oneskorenie. Bohužiaľ ani to nestačilo.

Predstavme si že sme stlačili klávesu. Klient odoslal informácie o vstupe serveru. Server odpovedal tým že sme videli ako sa naša postavička hýbe. Čas, ktorý ubehol od stlačenia klávesy po zobrazenie pohybu, však mohol prekračovať pol sekundy. To je na hru štýlu strieľačky príliš veľa. V dnešnej dobe optického internetu to nie je až taký problém. Vtedy to ale bol.

Client side predictions (predpovede na strane klienta). Keďže oneskorenie pôsobilo v multiplayerových hrách stále problém, vymyslel sa vylepšený model Klient/Server. Klient už nezobrazuje len to čo mu server odošle ako odozvu ale aj lokálne spracováva vstupy a na základe nich mení to čo hráč vidí hneď po stlačení klávesy. Znamenalo to, že klient musel spúšťať kód aj na vlastnom stroji.

Problém v tomto prístupe je riešenie konfliktov medzi dátami na servery a lokálnymi. Prečo by nemohol mať klient autoritu napríklad nad svojou polohou v hre? Pretože by to uľahčilo podvádzanie. Ak by sme mali totálnu autoritu nad svojou postavičkou v hre, nič by nám nebránilo povedať serveru že sa nachádzame napríklad za chrbtom nepriateľa a server by túto informáciu šíril ostatným klientom. Vo väčšine hier ma preto absolútnu autoritu server. V našej hre client side predictions neimplementujeme, pretože by to výrazne menilo našu hernú logiku.

3.3 Konklúzia

V našej hre je server autoritou nad všetkými objektami klientov s výnimkou postavičky. Tento spôsob umožňuje aby trieda postavičky mala rovnaký zdrojový kód pre server aj klienta. Hoci v našej hre existuje server a klient funkčnosťou sa podobá viac modelu peer to peer. Ak bude mať jeden hráč problémy so sieťou ovplyvní to aj ostatných, pretože ten čo má autoritu nad postavičkou rozposiela údaje o jej polohe ostatným. Preto sa môže stať že pohyb tejto postavičky bude pri slabom pripojení abnormálny.

V našej hre taktiež predpokladáme že užívateľ nebude skúšať podvádzať. Ako sme si už povedali, v modeli peer to peer je to jednoduché. Pretože implementujeme systém lobby, hráči sa najprv musia pripojiť k serveru a až tak začne hra. Počas tvorby tohto systému sa vyskytli rôzne výzvy, s ktorými sme sa museli popasovať. Na to aby bola hra hrateľná a zároveň nezaťažovala sieť sme museli použiť rôzne techniky. O synchronizácii pozície hráčov a dát ako takých pomocou Godot enginu a jeho nástrojoch si povieme v implementačnej časti.

4 Objasnenie základných pojmov

Pre pochopenie niektorých konceptov v ďalších častiach je potrebne objasniť si tieto pojmy. Povieme si prečo sme si zvolili práve Godot pre tvorbu našej hry a prečo je to neuveriteľne rýchlo rastúci a podporovaný engine.

4.1 Godot Engine

Pre tvorbu nášho projektu sme si vybrali Godot Engine. Jedným z dôvodov prečo sme si vybrali práve Godot je, že práca s ním je jednoduchá a jeho rozhranie je intuitívne. Godot je cross-platformový herný engine, ktorý umožňuje vytvárať 2D a 3D hry pomocou svojho rozhrania. Hry je možné exportovať na viacero hlavných desktopových platforiem (Linux, MacOS, Windows), ako aj mobilných (Android, iOS) a webových (HTML5) platforiem.

Godot je úplne zadarmo a open scource pod licenciou MIT. Hry používateľov sú ich, až na posledný riadok kódu enginu. Vývoj Godotu je úplne nezávislý a riadený komunitou, čo umožňuje používateľom pomáhať pri formovaní enginu tak, aby zodpovedal ich očakávaniam. Godot sa neustále vyvíja. Používatelia majú možnosť zdieľať svoje nápady a vylepšenia. Komunita je veľmi nápomocná. Doposiaľ sme neoľutovali že sme si vybrali práve tento engine.

4.2 GDScript

Náš projekt je kompletne napísaný v jazyku GDScript, preto si povieme o niektorých jeho vlastnostiach. GDScript je high level dynamicky písaný programovací jazyk. Používa syntax podobný Pythonu (bloky sú založené na odsadeniach a veľa kľúčových slov je podobných). Je optimalizovaný a pevne integrovaný s Godot enginom, čo umožňuje veľkú flexibilitu pre tvorbu obsahu a integráciu. Súbory majú príponu .gd. Musíme povedať že je veľmi príjemné s ním pracovať. Syntax je veľmi jednoduchý a po pár dňoch programovania sme si neuvedomovali že je to celkom nový programovací jazyk.

```gdscript
# Refresh player list and take care of name colors and avatars
func refresh_lobby():
    var players = Lobby.players
    player_list.clear()
    var item_idx = 0
    for p in players:
        if p == my_unique_id:
            player_list.add_item(players[p][0] + ' (You)')
            my_item_list_idx = item_idx
        else:
            player_list.add_item(players[p][0])
        if players[p][1]:
            player_list.set_item_custom_fg_color(item_idx, Color(0.0, 1.0, 0.0))
        else:
            player_list.set_item_custom_fg_color(item_idx, Color(1.0, 0.0, 0.0))
        item_idx += 1
        avatar_texture.set_texture(load('res://Sprites/ships/' + players[my_unique_id][2]))
```

Obr. 2 Funkcia refresh_lobby - ukážka syntaxu GDScript (Zdroj: JumpAround 2019)

4.3 Uzly

Pri popise implementácie nášho projektu budeme hovoriť o Uzloch (Node). Uzly sú základnými stavebnými kameňmi na vytvorenie hry. Teoreticky by sa dali uzly predstaviť ako objekty v programovaní. Každý uzol je objekt, no nie každý objekt je uzol. Každý uzol má tieto atribúty:

- názov
- upraviteľné vlastnosti
- môže spracovávať funkciu každý frame
- môže byť rozšírený
- može byť priradený inému uzlu ako dieťa

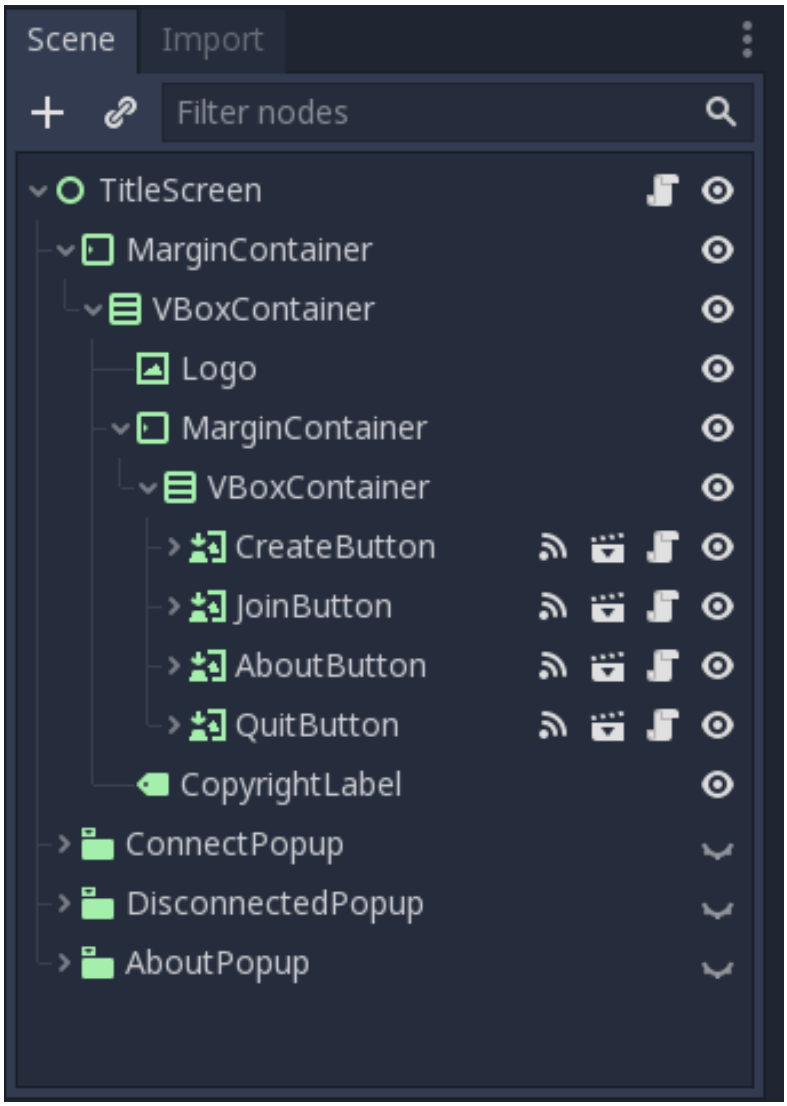
Keďže uzly môžu mať iné uzly ako deti z ich usporiadania sa stáva stromom. Schopnosť usporiadať uzly týmto spôsobom je veľmi silným nástrojom. Godot má obrovské množstvo uzlov s vlastnosťami a funkciami potrebnými pri tvorbe hier. Používateľ môže tvoriť uzly, ktoré dedia z uzlov už pred tvorených. Nič mu ale nebráni v tom aby si vytvoril vlastné uzly a implementoval v nich jeho vlastnú hernú logiku. To presne sme urobili aj mi v našom projekte. O našich uzloch si povieme v implementačnej kapitole.

4.4 Scény

Scéna sa skladá zo skupiny hierarchicky usporiadaných uzlov (v stromovom modeli). Spustenie hry znamená spustenie scény. Projekt môže obsahovať niekoľko scén, ale aby hra začala, jeden z nich musí byť vybraný ako hlavná scéna. Súbory scén majú v Godote príponu .tscn. Náš projekt obsahuje množstvo scén, o ktorých si povieme v implementačnej časti.

Scéna vždy:

- jeden koreňový uzol
- schopnosť uloženia na disk a neskôr znova načítania
- môže byť inštanciovaná



Obr. 3 Scéna a usporiadanie uzlov do stromu (Zdroj: Projekt JumpAround 2019)

4.5 Signály

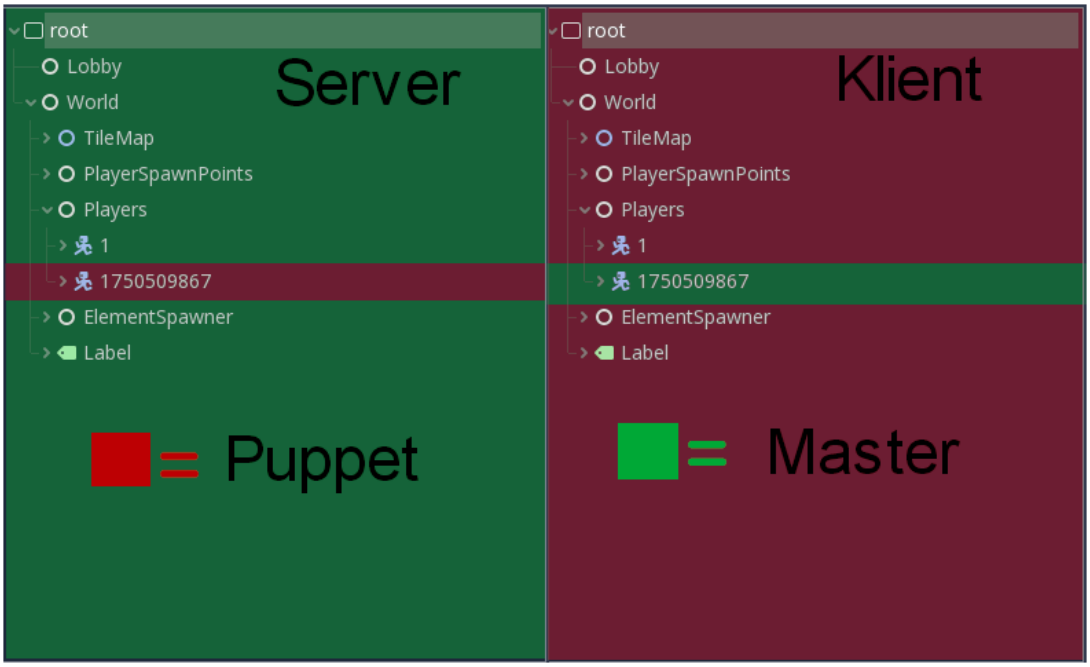
Signály umožňujú uzlu aby odoslal správu, ktorú môžu ostatné uzly očakávať a reagovať na ňu. Namiesto toho aby sme stále kontrolovali či je tlačidlo stlačené, naprogramujeme ho tak aby vyslal signál po jeho stlačení. Umožňuje to lepšiu efektivitu a organizáciu kódu. Signály sú naozaj mocný nastroj pri tvorení hier. Každý uzol má vlastný počet signálov, ktoré sa vyšlú za určitých podmienok. V našom projekte využívame množstvo signálov a máme vytvorených aj niekoľko vlastných.

5 Implementácia

V tejto časti si vysvetlíme jednotlivé scény a ich pripojené skripty. Pri každej scéne si popíšme jej vlastnosti a funkcie. Povieme si to tom aký je ich účel a ako spolu interagujú . Každá scéna má vizuálnu zložku (.tscn) a logickú zložku (.gd). Vizuálna zložka sa stará o to čo užívateľ vidí, zatiaľ čo logická sa zaoberá programovou logikou. Povieme si ako sme synchronizovali pohyby hráčov pre jednotlivých užívateľov. Na koniec si povieme, v čom sme kreslili obrázky pre našu hru.

5.1 Synchronizácia hráčov

V Godote vystupuje ten kto sa inicializoval serverom ako Master, a ten, kto sa pripojil na server Puppet. Tento model ale však nie je premenovaným Klient/Serverom. Hoci sú podobné. Sieťový master uzla je ten kto má nad ním absolútnu autoritu. Ak to nie je explicitne nastavené, sieťový master bude vždy server(network_id: 1). Keďže naša implementácia synchronizácie sa viac podobá na peer to peer, chceme aby bol network mastrom každý svojej vlastnej postavičky. Preto každej inštancii scény hráča nastavíme ako network_mastra jej peera ktorému patrí,



Obr. 4 Strom scény počas hry - zobrazenie modelu Master/puppet (Zdroj: JumpAround 2019)

5.1.1 RPC

Na komunikáciu medzi peermi sa používajú RPC(Remote Procedure Call). Implementácia funguje na základe týchto metód:

- rpc(„názov funkcie“, voliteľné argumenty) - broadcast funkcie, ktorú majú peeri zavolať
- rpc_id(peer_id, „názov funkcie“, voliteľné argumenty) - poslanie funkcie iba jednému peerovi
- rpc_unreliable(„názov funkcie“, voliteľné argumenty) - to isté ako rpc(), ale doručenie nie je zaistené
- rpc_unreliable_id(peer_id, „názov funkcie“, voliteľné argumenty)
- Na synchronizáciu premenných sa používa metóda rset(). Tú ale v našom projekte nepoužívame.

5.1.2 RPC kľúčové slová

Tieto kľúčové slová majú dve využitia. Prvé je, že Godot bude vedieť že daná metóda bude volaná prostredníctvom RPC. Ak nie sú použité žiadne, volanie prostredníctvom RPC bude blokované kvôli bezpečnosti. Druhé je špecifikovanie ako bude volanie prebiehať.

- remote – volanie vykonajú všetci ostatní peeri
- remotesync – volanie vykonajú všetci ostatní peeri ale zároveň zavoláme metódu aj lokálne
- master – iba network máster zavolá metódu
- puppet – iba network puppet zavolá metódu

5.1.3 RPC v našej hre

Čo to vlastne znamená ? Ak v našej hre nejaký peer detekuje že jeho postavička mala dostať zásah zavolá metódu rpc(„hit“). Metóda hit() má pred sebou kľúčové slovo remotesync. To znamená že ju vykonajú všetci peeri a zároveň aj ten, ktorý volal RPC,

v našom prípade network_master postavičky. Podobne to funguje aj s ostatnými metódami. Napríklad: remotesync func add_coin() alebo remotesync func speed_up().

5.1.4 Synchronizácia pozície

Na to aby sme synchronizovali pozície hráčov pre všetkých peerov sme použili nasledujúci spôsob. Každý network_master postavičky rozposiela ostatným rpc(„update_pos“, pos) približne v intervaloch 30ms. V tejto metóde (remote func update_pos(updated_position)) každý peer počas 40ms interpoluje pozíciu postavičky na novú pozíciu. Týmto sa zaručí hladký pohyb.

5.2 Lobby.gd

Lobby je hlavný skript, ktorý sa stará o správu spojenia hráčov a ovláda priechod hry.

Je automaticky načítaný po spustení programu(autoloaded). Reaguje na signály pripojenia a odpojenia peerov a určuje, ktoré rozhranie sa aktuálne zobrazuje. Najdôležitejšie metódy zahrňujú:

- exit_lobby(abnormaly) – slúži na zrušenie pripojenia na server a vrátenie sa na TitleScreen, a ak sa abnormaly = true znamená to, že spojenie zlyhalo
- change_avatar(id) – použité pri zmene farby lode v LobbyScreen
- change_player_name(new_name, id) – slúži na zmenenie mena v LobbyScreen
- create_server() – metóda na vytvorenie servera
- connect_to_server(ip) - metóda na pripojenie na server
- update_puppets(updated_players) – metóda na informovanie ostatných peerov o novom hráčovi
- _player_disconnected(id) – metóda volaná pri odpojení jedného z peerov
- _server_disconnected() - metóda volaná pri odpojení servera
- set_spawn_points() – metóda, ktorú volá server aby priradil hráčom náhodné začiatočné pozície
- load_game() – slúži na načítanie hry
- start_round() – slúži na spustenie kola
- check_end_round() – kontroluje či je možné určiť víťaza a ukončiť kolo
- end_game() – po ukončení všetkých kôl ukončí hru

5.3 Player.tscn

Player, je scéna, ktorá slúži primárne ako postavička hráča. Každý hráč ovláda práve jednu inštanciu tejto scény. Je to najzložitejšia scéna z celého projektu. Vysvetlíme si aké sú jej súčasti. Hlavný uzol má v stromovej hierarchii ako deti tieto uzly:

- WinningLabel – Predstavuje text v tvare „Winner“. Počas hry je skrytá, zobrazí sa na určitý čas len nad tým hráčom, ktorý vyhral kolo.
- ShipSprite – Predstavuje obrázok lode hráča. Môže nadobúdať rôzne farby podľa výberu v lobby. Počas hry sa otáča a orientuje podľa smeru pohybu hráča.
- ShieldSprite – Predstavuje obrázok energetického štítu. Prekrýva obrázok lodi hráča vtedy, ak nemôže byť poškodený od Damagera.
- StatSprite – Spoločne so StatLabel zobrazujú počet zdravia hráča. Sú viditeľné len vtedy ak hráč život stratil alebo zobral Orb.
- EngineParticles – Častice, ktoré vyžarujú spoza obrázka lode, ak je vo vzduchu. Pri páde sa tieto časti vypnú.
- JumpParticles – Častice, ktoré sa vyžarujú po skoku hráča.
- HitParticles – Častice, ktoré sa vyžarujú vtedy, keď je hráč zasiahnutý. Prestanú sa vyžarovať po tom ako hráč znovu získa kontrolu nad svojou postavičkou.
- WinningParticles – Častice, ktoré sa na hráčovi vyžarujú iba pri výhre kola podobne ako WinningLabel.
- WorldCollision – Dvojrozmerná oblasť, ktorá slúži na detekciu kolízie s okolitým svetom (zem, strop, steny, elementy).
- DownTimer – Časovač, slúžiaci na odpočítanie času, po uplynutí ktorého zavolá metódu na povoleniu skoku hráča.
- StatTimer – Tento časovač slúži na to, aby sa istý čas zobrazil StatSprite spoločne so StatLabel a upozornili hráča na jeho nový stav životov.
- ShieldTimer – Slúži na odpočítanie času, počas ktorého hráč vidí na svojej postavičke ShieldSprite.
- ImmortalTimer – Počas trvanie tohto časovača je hráč nezraniteľný ostatnými hráčmi.

- KnockoutArea – Dvojrozmerná oblasť, slúžiaca na detekciu kolízie s ostatnými hráčmi.
- Tween – Uzol, pomocou ktorého hráč interpoluje svoju pozíciu získanú od network mastra.

Pohyb hráča je implementovaný vo funkcii _physics_process(delta). Táto funkcia je volaná 60-krát za sekundu v rovnakých intervaloch. Delta – ubehnutý čas od predchádzajúceho volania by preto mala byť približne 0,016666 sekundy. Pomocou premenných: SPEED, GRAVITY, JUMPHEIGHT, motion, jump_index, motion a mnohých ďalších je zabezpečený pohyb a spôsob interagovania so svetom.

Vstup (dotyk obrazovky) je implementovaný v metóde _input(event), v ktorej event vyjadruje typ vstupu.

5.4 Element.tscn

Scéna môže dediť z inej scény, presne tak ako jeden objekt z druhého v objektovom programovaní. Presne túto možnosť využívame pri elementoch. Scény Damager, Reverser, SpeedUp a Orb všetky dedia zo scény Element. Za elementy považujeme objekty, s ktorými môže hráč počas hrania interagovať vo svete.

Každý element je tvorený (spawnovaný) na náhodnej x-ovej súradnici a mimo obrazovky na y-ovej súradnici. Následne sa elementy hýbu smerom dole, resp. hore a po prejdení sa vymažú. Viac o tvorení elementov si povieme v ďalšej časti. Elementy obsahujú:

- Sprite – Obrázok pre každý individuálny element.
- AnimationPlayer – Uzol, vďaka ktorému prehráme určitú animáciu na elemente.
- CollisionShape2D – 2D oblasť pre detekciu kolízie s elementami.

5.4.1 Damager.tscn

Damager je element, ktorého úlohou je zraniť hráča. Po kontakte s ním zavolá metódu hráča hit(). Táto metóda hráčovi odoberie život a znemožní akýkoľvek ďalší

pohyb. Po čase však hráč znovu získa schopnosť pohybu. Damager na rozdiel od iných elementov počas svojho pohybu rotuje.

5.4.2 Reverser.tscn

Úlohou Reversera je otočiť hráča po kontakte s ním. To dosahuje tým, že zavolá metódu flip(), ktorá hráčovi zmení jeho smer pohybu a obráti obrázok. Pri tomto volaní však zároveň vytvorí nový časovač. Počas priebehu tohto časovača hráč nebude môcť znovu interagovať resp. odraziť sa od Reversera.

5.4.3 SpeedUp.tscn

Pomocou tohto elementu si vie hráč dočasne zvýšiť rýchlosť pohybu svojej postavičky. Po kontakte element zavolá metódu speed_up(). Táto metóda zdvojnásobí rýchlosť postavičky a upraví EngineParticles aby si hráč všimol ako zrýchlil. S touto metódou taktiež spustí časovač. Po uplynutí času zavolá metódu speed_body_down() a hráč bude mať opäť svoju pôvodnú rýchlosť.

5.4.4 Orb.tscn

Orb má dve významy. Prvý je taký, že slúži ako bod pri určovaní víťaza po odohraní všetkých kôl. Druhý je v kole samotnom. Ak hráč nazbiera tri Orby, pripočíta si tak zdravie. O toto všetko sa stará metóda add_coin(), ktorú Orb zavolá po dotyku s hráčom. Orb sa taktiež po dotyku vymaže.

5.5 ElementSpawner.gd

Popri tom ako sme si hovorili o elementoch určite stojí za to povedať si aj o ElementSpawneri a jeho funkcionalite. Je to skript priradený uzlu v scéne World. Slúži na to, aby sme kontrolovane vytvárali elementy pomocou určitých metód. Vytváranie začíname spustením metódy start_spawn(). Túto metódu spúšťa iba server. V nej sa vytvorí časovač. Po vypršaní času pomocou signálu zavolá metódu spawn(). Časovač sa potom resetuje a znovu začne odpočítať. Tak dosiahneme že zavolá metódu napríklad každé štyri sekundy.

Metóda spawn() náhodne generuje tri parametre, s ktorými následne zavolá metódu spawn_element() na všetkých pripojených počítačoch pomocou RPC. Tieto parametre sú:

- element_id(číslo elementu označujúce či sa jedná o Damager, Reverser...)
- element_orientation(boolean hovoriaci o tom či sa element objaví hore alebo dole)
- element_position(značí x-ovú pozíciu elementu vo svete)

5.6 World.tscn

World je scéna, v ktorej sa odohráva hra. Táto scéne nemá k sebe priradený žiadny skript. Je to iba pozadie hry, ohraničenie hernej plochy a kontajner pre všetky uzly hry. Skladá sa z týchto uzlov:

- TileMap – Toto je scéna sama o sebe obsahujúca iba 4 kolízne objekty predstavujúce steny, zem a strop spoločne s pozadím hry.
- PlayerSpawnPoints – Uzol vytvorený len na to, aby slúžil ako kontajner pre šesť 2D súradníc určených ako začiatočné pozície pre hráčov.
- Players – Uzol vytvorený ako kontajner pre inštancie scény Player po začiatku hry.
- Label – Text, ktorý sa objaví po štarte hry a upozorní hráčov aby sa pripravili.

5.7 Užívateľské rozhrania

Pre interakciu užívateľa s programom sme vytvorili tri rôzne rozhrania. Každé bolo vytvorené s úmyslom byť prehľadné, jednoduché a intuitívne. Všetky rozhrania sú usporiadané v boxových štruktúrach.

5.7.1 TitleScreen.tscn

Toto rozhranie privíta užívateľa hneď po spustení programu. Je to základným rozhraním, z ktorého sa vytvára lobby alebo pripája do už vytvoreného. Obsahuje:

- Logo
- Tlačidlo na vytvorenie hry. Po stlačení bude užívateľovi zobrazené rozhranie LobbyScreen

- Tlačidlo na pripojenie do hry. Po stlačení sa užívateľovi zobrazí popup okno do ktorého musí zadať IP adresu servera.
- Tlačidlo na prečítanie si pravidiel hry.
- Tlačidlo na vypnutie programu
- Text s menom autora programu

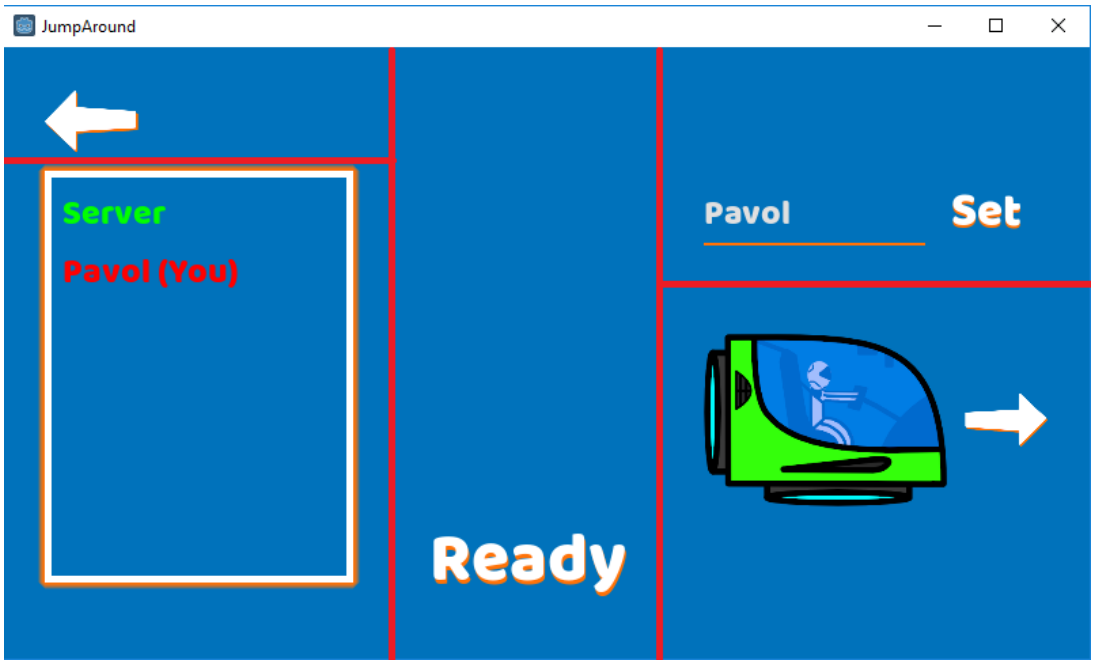
Obr. 5 TitleScreen rozhranie a usporiadanie uzlov v boxovej štruktúre (Zdroj: JumpAround 2019)

5.7.2 LobbyScreen.tscn

Po vytvorení hry, alebo po pripojení do hry sa užívateľ dostane do tohto rozhrania. Užívateľ môže v lobby

- Zmeniť farbu lode pomocou tlačidla po pravej strane
- Zmeniť meno napísaním do riadku a následným stlačením tlačidla Set
- Vidieť mená všetkých pripojených hráčov a to či sú pripravený na štart hry (Server je automaticky pripravený)
- Zmeniť svoj stav pripravenosti

Na to aby server mohol spustiť hru musí každý pripojený hráč stlačiť tlačidlo Ready. Jedine server má možnosť stlačiť tlačidlo Start, ktorým spúšťa hru.



Obr. 6 LobbyScreen rozhranie a usporiadanie uzlov v boxovej štruktúre (Zdroj: JumpAround 2019)

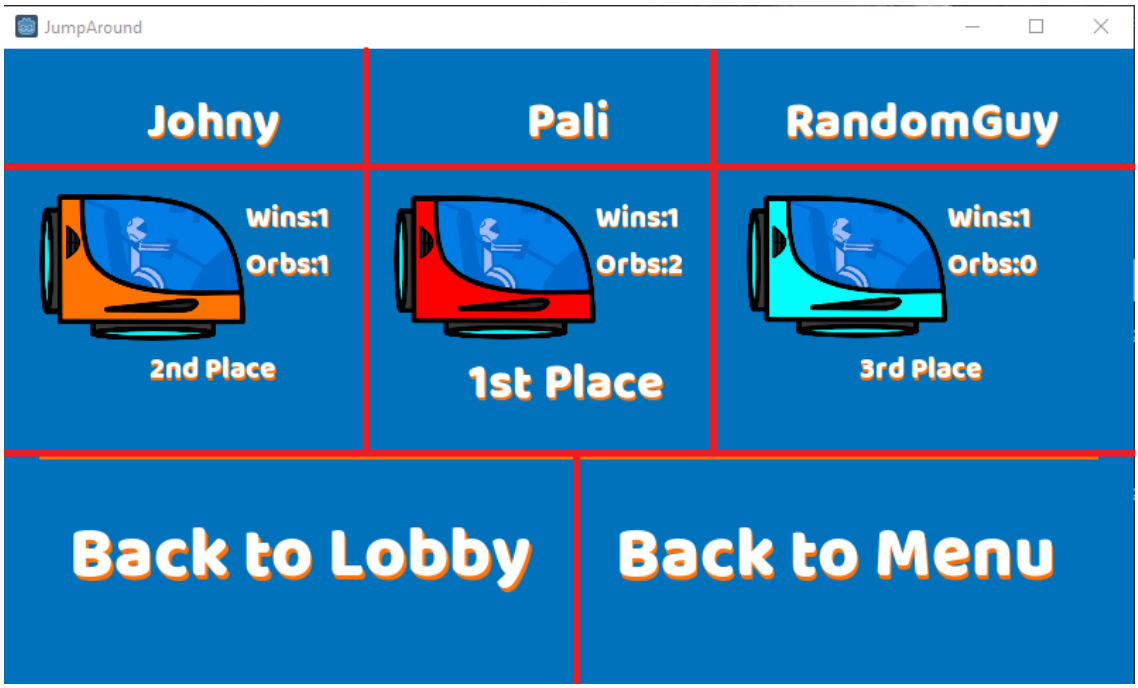
5.7.3 ResultScreen.tscn

ResultScreen predstavuje rozhranie, v ktorom si užívateľ môže pozrieť výsledky z hry. Ak hru hrali traja a viac hráčov, zobrazia sa prvé tri miesta. Ak hrali iba dvaja, zobrazia sa iba dve pozície. Užívateľ si môže pozrieť:

- Mená hráčov
- Počet výhier jednotlivých hráčov
- Počet nazbieraných Orbov

Užívateľ ma následne dve možnosti:

- Tlačidlom Back to Lobby sa vráti spať na rozhranie LobbyScreen
- Tlačidlom Back to Menu sa vráti na rozhranie TitleScreen



Obr. 7 ResultScreen rozhranie a usporiadanie uzlov v boxovej štruktúre (Zdroj: JumpAround 2019)

5.7.4 MenuButton.tscn

Všetky textové tlačidla sú inštancie tejto scény. Tento spôsob uľahčuje prípadne zmenenie štýlu tlačidiel. V Godote neexistuje funkcionalita na zmenenie farby fontu tlačidla. Táto scéna preto obsahuje tlačidlo a za svoje dieťa má textový uzol. Vďaka tomu získame funkcionalitu zmenenia farby fontu po stlačení.

5.8 Použitý font

Vo všetkých našich textových uzloch je použitý jediný font. Volá sa Baloo a je pod OFL(Open Font License) licenciou. Je používaný vo veľkostiach 34, 56, 82. Vybrali sme si ho pretože dodáva rozhraniu jedinečný uhladený vzhľad.

5.9 Krita

Pri tvorbe našej hry sme potrebovali na tvorbu grafiky zvoliť program, v ktorom bude kreslenie a upravovanie obrázkov jednoduché a intuitívne. Vybrali sme si program Krita, ktorý je podobne ako Godot MIT licencovaný. Obrázky sme si rozdelili do mnohých vrstiev

pre lepšiu prehľadnosť a manipuláciu. Všetky obrázky použité hre sú vytvorené nami v tomto programe.

Zhrnutie

Našim cieľom bolo vytvoriť hru pre viacerých hráčov na platformu Android. Komunikácia zariadení mála prebiehať prostredníctvom lokálnej siete. Hru by tak mohli hrať priatelia bez toho, aby museli sedieť pri jednom počítači. V prvom rade sme sa museli rozhodnúť, aký herný engine a či vôbec použijeme. Následne sme sa museli rozhodnúť pre program, v ktorom budeme tvoriť grafiku. Potrebovali sme vedieť ako fungujú protokoly TCP a UDP, aby sme sa rozhodli, ktorý v našej hre použijeme. Naučili sme sa, ako sa herní vývojári pasovali so synchronizáciou hier na viacerých zariadeniach.

Pre tvorbu našej hry sme si zvolili herný engine Godot. Spoločne s programom Krita sme mali všetky potrebné nástroje na tvorbu plnohodnotnej hry. Ďalšou výzvou bolo, osvojiť si techniky tvorby hier a kreslenia grafiky. Museli sme precízne preštudovať rôzne koncepty. Pri synchronizácii hry to boli modely peer to peer a klient/server. Pri tvorbe hernej logiky sme zase rozoberali pojmy ako uzol, scéna alebo signály. Museli sme sa naučiť pracovať s celkom novým programovacím jazykom. Po pochopení všetkých týchto konceptov sme však prišli na to, že tvorba plnohodnotnej hry je v dnešnej dobe dostupná pre každého a vytvoriť plnohodnotnú hru zvládne aj neskúsený nováčik.

Resumé

Our goal was to create a multiplayer game on the Android platform. Devices would communicate through the local network. This way the players would not have to sit behind one computer just to play. First and foremost, we had to choose which game engine, if any we would be using. Subsequently, we needed to pick a program in which we could create graphics for our game. In order to pick the right protocol for our game communication, we had the learn what are the differences between UDP and TCP. We had learned how programmers dealt with synchronizing data across multiple devices.

Our game engine of choice became Godot. Together with the program Krita, we had everything we needed to create a full-featured game. Learning game making and drawing techniques was another challenge we had to overcome. We had to study precisely various concepts. Peer to peer and client / server models were important to understand in order to program a properly synchronized game. While explaining game logic, we discussed terms such as node, scene or signals. We had to learn to work with a whole new programming language. After understanding all these concepts, we have learned that creating a full-featured game can be done, even by an inexperienced novice.

Zoznam použitej literatúry

Glenn Fiedler: UDP vs. TCP: Which protocol is best for games? https://gafferongames.com/post/udp_vs_tcp/ (2004-2019)

Glenn Fiedler: A short history of game networking techniques https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/ (2004-2019)

Juan Linietsky, Ariel Manzur and the Godot community: High level multiplayer. https://docs.godotengine.org/en/3.1/tutorials/networking/high_level_multiplayer.html (2014-2019)

Juan Linietsky, Ariel Manzur and the Godot community: About Godot Engine. https://docs.godotengine.org/en/3.1/about/introduction.html (2014-2019)

Juan Linietsky, Ariel Manzur and the Godot community: Scenes and nodes. https://docs.godotengine.org/en/3.1/getting_started/step_by_step/scenes_and_nodes.html (2014-2019)

Juan Linietsky, Ariel Manzur and the Godot community: Signals. https://docs.godotengine.org/en/3.1/getting_started/step_by_step/signals.html (2014-2019)