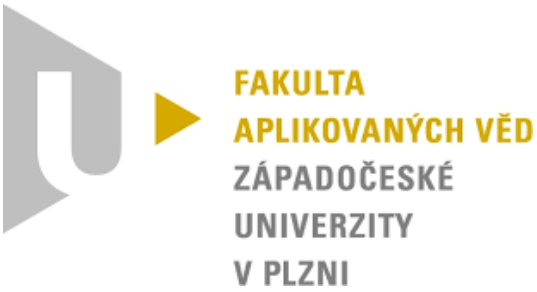

Semestrální práce z předmětu KIV/UPS

Realtime online multiplayer hra - Asteroidy

Patrik Janoušek

A17B0231P
janopa@students.zcu.cz

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Obsah

1 Popis hry

Vypracovaná hra Asteroidy je velmi inspirována originální hrou Asteroids z roku 1979. Avšak je převzata spíše původní myšlenka hry, na kterou jsou aplikovány vlastní nápady a úpravy.

Jedná se o hru s velmi primitivními pravidly. Hráč ovládá vesmírnout loď, se kterou má možnost střílet do okolních asteroidů nebo hráčů. Zároveň může umřít za předpokladu, že ho sestřelí jiný hráč, nebo se středne s letícím asteroidem. V takovém případě je hráč přesunut na jinou pozici a je mu udělena tříSekundová imunita. Po dobu trvání této imunity je mu samozřejmě znemožněno střílet, aby nebyl zvýhodněn nad ostatními hráči. Za smrt se neuděluje žádná penalizace. Za zabití protihráče je 1 000 bodů, za asteroid je to pak 1-100 bodů podle jeho velikosti. Hra končí ve chvíli, kdy nějaký hráč dosáhne 10 000 bodů.

Obrázek 1: Snímek z vypracované hry asteroidy

2 Protokol

2.1 Specifikace

Protokol byl navržen s ohledem na jednoduchost implementace, ale aby zároveň splnil všechny požadavky, které by aplikace mohla mít.

Mezi hlavní požadavky na protokol byla identifikace typu zprávy. Tu je možné provádět pomocí trojciferných číselných identifikátorů. Díky těmto identifikátorů je schopná komunikující protistrana předpokládat strukturu příchozích dat.

Podporuje tedy identifikaci zpráv, kde si klient může určit vlastní identifikátor zprávy, a server tento identifikátor zapíše i do odpovědi, díky čemuž je klient schopen poznat na jakou zprávu server odpovídá. Tento identifikátor má variabilní délku, tudíž je zde i možnost jeho nepoužití.

Tělo zprávy je pak zakódované ve formátu JSON, kde je pro klíče použita konvence snake case.

Popis	Délka	Hodnota
Dělící znak	1	„|“
Typ zprávy	3	Číslo v ASCII
Dělící znak	1	„|“
Délka těla zprávy	variabilní	Číslo v ASCII
Dělící znak	1	„|“
Identifikátor požadavku	variabilní	Řetězec znaků a-z, A-Z a 0-9 v ASCII
Dělící znak	1	„|“
Tělo zprávy	variabilní	Data zakódovaná v JSONu

Tabulka 1: Specifikace protokolu

3 Implementace

3.1 Popis hry

3.2 Definice použitých zpráv

Jak již bylo uvedeno výše, každý typ zprávy má svůj unikátní trojciferný identifikátor. Tento identifikátor může být z hlediska protokolu libovolný, nicméně pro lepší orientaci byla v konkrétní implementaci vytvořena určitá pravidla.

A to taková, že identifikátor 1xx je použit pro obecné zprávy, které nemusí přímo souviset se samotnou aplikací (keep-alive, atd. . .). 2xx pak slouží pro zprávy zasílané klientem, a 3xx zprávy zasínalé serverem. Zde je ještě dodržováno to, že „xx“ je stejné pro klientskou i serverovou zprávu, pokud je serverová zpráva odpovědí na klientský požadavek. Tyto identifikátory jsou použity pro zprávy, které přímo nesouvisí se hrou, ale spíše s její správou (autentizace, založení lobby, získání seznamu hráčů, . . .). Nakonec jsou pak

použity identifikátory zpráv 4xx a 5xx, kde 4xx jsou zprávy zasílané klientem, a 5xx zprávy zasílané serverem. Tyto identifikátory jsou použity pro zprávy přímo související se hrou (pohyb hráče po herní ploše, střelba, ...).

Všechny zprávy zasílané serverem, jsou obalovány do struktury, která umožňuje k zasílaným datům přidat status, a případně nějakou dodatečnou zprávu. Přenášená zpráva je pak dostupná pod klíčem „data“.

Klíč	Datový typ
data	object
message	string
status	boolean

Tabulka 2: Obalovací zpráva serveru

3.2.1 Zprávy zasílané klientem

KeepAlive

Typ zprávy: 100

Tato zpráva je zasílána klientem, aby ověřila funkčnost spojení se serverem, a zároveň schopnost komunikace mezi těmito dvěmi stranamy.

Klíč	Datový typ
ping	string

Tabulka 3: Specifikace KeepAlive

Klient jako hodnotu pole „ping“ vyplňuje řetězec „pong“. A server na tuto zprávu odpovídá stejným typem zprávy, jen jako hodnotu pole „ping“ vyplňuje řetězec „ping-pong“.

ActionError

Typ zprávy: 101

Zpráva, která je zasílána serverem jako odpověď na typ zprávy, ke které klient nemá oprávnění, nebo daná zpráva neexistuje. Tato zprávy má prázdné tělo a neposílá žádná dodatečná data.

Authenticate

Typ zprávy: 200

Autentizační zpráva musí být klientem zaslána jako první zpráva po připojení k serveru. V případě neposlání této zprávy nemá klient žádné pravomoce, mimo zasílání *KeepAlive*.

Klíč	Datový typ	Popis
name	string	Jméno hráče

Tabulka 4: Specifikace Authenticate

CreateLobby

Typ zprávy: 201

Zpráva určená k založení lobby. Hráč, který lobby vytváří, je automaticky do lobby připojen a není již potřeba posílat *JoinLobby*.

Klíč	Datový typ	Popis
name	string	Název lobby
players_limit	integer	Limit počtu hráčů v lobby

Tabulka 5: Specifikace Authenticate

DeleteLobby

Typ zprávy: 202

Zpráva určená ke smazání lobby. Ačkoliv zpráva přjímá jako parametr název lobby ke smazání, tak hráč má právo smazat lobby jenom za předpokladu, že je jejím vlastníkem.

Klíč	Datový typ	Popis
name	string	Název lobby ke smazání

Tabulka 6: Specifikace DeleteLobby

ListLobbies

Typ zprávy: 203

Prázdná zpráva sloužící k získání seznamu všech dostupných lobby.

JoinLobby

Typ zprávy: 204

Zpráva sloužící k připojení do lobby. Připojení je možné pouze za předpokladu, že je v lobby volný slot pro hráče. Tato skutečnost je ověřována serverem, který vyplní patřičný status zprávy.

Klíč	Datový typ	Popis
name	string	Název lobby k připojení

Tabulka 7: Specifikace JoinLobby

LeaveLobby

Typ zprávy: 214

Prázdná zpráva, kterou se hráč odpojí z lobby.

ListLobbyPlayers

Typ zprávy: 206

Prázdná zpráva sloužící k získání seznamu hráčů přípojených do lobby.

StartGame

Typ zprávy: 207

Prázdná zpráva pro spuštění hry. Tuto zprávu může poslat libovolný hráč z lobby. Od této chvíle je na serveru vytvořená instance hry, a všichni hráči jsou neustále informováni o jejím stavu.

GameReconnectAvailable

Typ zprávy: 211

Touto prázdnou zprávou si může klient ověřit, zda připojený hráč nepatří do nějaké hry, a tudíž se do ní může opět připojit, nebo se z ní odpojit.

Reconnect

Typ zprávy: 212

Zasláním této prázdné zprávy se hráč připojí zpět do probíhající hry, kterou předtím opustil.

LeaveGame

Typ zprávy: 213

Zasláním této prázdné zprávy hráč definitivně opustí probíhající hru, ze které odešel, a již se do ní nemůže vrátit.

PlayerMove

Typ zprávy: 400

Zpráva, která informuje server o změně atributů hráče. Tuto zprávu by měl klient posílat při změně své pozice, nebo libovolného atributu svého pohybu (vektor pohybu, rotace, ...). Zaslané údaje jsou na serveru verifikovány a v případě chybně zaslaných údajů je klient vyžádán k synchronizaci údajů se serverem (viz. *UpdateState*).

Klíč	Datový typ	Popis
pos_x	double	Pozice na ose X
pos_y	double	Pozice na ose Y
velocity_x	double	Složka X vektoru pohybu
velocity_y	double	Složka Y vektoru pohybu
rotation	double	Rotace hráče v radiánech

Tabulka 8: Specifikace PlayerMove

ShootProjectile

Typ zprávy: 401

Prázdná zpráva informující server o vystřelení projektilu

3.2.2 Zprávy zasílané serverem

CreatedLobbyResponse

Typ zprávy: 301
Odpověď na zprávu: *CreatedLobbyResponse*

Prázdná zpráva, kterou je klient informován o založení lobby.

DeleteLobbyResponse

Typ zprávy: 302

Prázdná zpráva, kterou je klient informován o smazání lobby.

ListLobbiesResponse

Typ zprávy: 303
Odpověď na zprávu: *ListLobbies*

Tuto zprávu server používá pro zaslání seznamu všech dostupných lobby. V kořenu zprávy je jedinný klíč „lobbies“, který má následující strukturu:

Klíč	Datový typ	Popis
name	string	Název lobby
connected_players	integer	Počet hráčů připojených do lobby
players_limit	integer	Maximální počet připojených hráčů

Tabulka 9: Specifikace ListLobbiesResponse

JoinLobbyResponse

Typ zprávy: 304
Odpověď na zprávu: *JoinLobby*

Prázdná zpráva informující klienta o stavu připojení do lobby.

PlayerLobbyJoined

Typ zprávy: 305

Zpráva informující hráče, kteří jsou již připojení v lobby, o připojení nového hráče.

Klíč	Datový typ	Popis
name	string	Jméno hráče

Tabulka 10: Specifikace ListLobbiesResponse

ListLobbyPlayersResponse

Typ zprávy: 306
Odpověď na zprávu: *ListLobbyPlayers*

Zpráva, kterou server klientovi vrací seznam hráčů připojených do lobby.

Klíč	Datový typ	Popis
players	string[]	Pole jmen hráčů

Tabulka 11: Specifikace ListLobbyPlayersResponse

StartGameResponse

Typ zprávy: 307
Odpověď na zprávu: *StartGame*

Prázdná zpráva, která se zasílá všem hráčům při startu hry.

LobbyPlayerConnected

Typ zprávy: 308

Zpráva informující ostatní hráče v lobby o připojení nového hráče.

Klíč	Datový typ	Popis
name	string	Jméno příchozího hráče

Tabulka 12: Specifikace LobbyPlayerConnected

LobbyPlayerDisconnected

Typ zprávy: 309

Zpráva informující ostatní hráče v lobby o odpojení hráče.

Klíč	Datový typ	Popis
name	string	Jméno odpojeného hráče

Tabulka 13: Specifikace LobbyPlayerDisconnected

GameEnd

Typ zprávy: 310

Tato zpráva je zaslána všem hráčům hry po jejím skončení. V kořenu zprávy obsahuje jediný klíč „score_summary“, který obsahuje pole s objekty následující struktury:

Klíč	Datový typ	Popis
player_name	string	Jméno hráče
score	integer	Dosažené skóre
winner	boolean	Informaci o tom, zda je daný hráč výherce

Tabulka 14: Specifikace GameEnd

GameReconnectAvailableResponse

Typ zprávy: 311
Odpověď na zprávu: GameReconnectAvailable

Zpráva informující o tom, zda je možné se připojit zpět do probíhající hry, ze které se hráč odpojil. Obsahuje pouze jednou hodnotu:

Klíč	Datový typ	Popis
available	boolean	Dostupnost znovupřipojení se do probíhající hry

Tabulka 15: Specifikace GameReconnectAvailableResponse

ReconnectResponse

Typ zprávy: 312
Odpověď na zprávu: Reconnect

Prázdná zpráva informující klienta o úspěšném znovupřipojení zpět do hry.

LeaveGameResponse

Typ zprávy: 313
Odpověď na zprávu: LeaveGame

Prázdná zpráva informující klienta o úspěšném opuštění probíhající hry.

LeaveLobbyResponse

Typ zprávy: 314
Odpověď na zprávu: LeaveLobby

Prázdná zpráva informující klienta o úspěšném opuštění lobby.

3.3 Server

Pro implementaci serveru byl zvolen jazyk go. Jednak z důvodu vyzkoušení nové technologie, ale i z důvodu jeho technologických výhod. Zejména z důvodu jednoduchosti paralelizace a možnosti jednoduché a bezpečné komunikace mezi jednotlivými vlákny (resp. gorutinami), ale i automatické správy paměti.

3.3.1 Síť

Síť je v serveru implementována s důrazem na co největší oddělení od zbytku aplikace, a je vedle aplikace v samostatném balíku „net“, který je pojmenován po vzoru stejnojmenného balíku ze standardní knihovny jazyka go. Tento balík obsahuje implementaci jak samotného TCP serveru, tak i protokolu.

Protokol je plně oddělen od TCP serveru, se kterým komunikuje pomocí rour. Díky tomu je protokol schopen komunikovat s libovolným binárním streamem dat, který nutně nemusí přijít od TCP serveru. Tato vlastnost ho dělá i samostatně testovatelným. Nevýhodou však je, že je díky tomu komunikace s protokolem blokující, a každý klient musí mít pro protokol vlastní gorutinu.

Pokud tedy přijde zpráva, tak jí TCP server přečte ze socketu, a zapíše jí do patřičné roury připojeného klienta, odkud zprávu začne přečte protokol, a zečne jí zpracovávat. Po tom, co protokol přečte celou zprávu, tak ji naplní do struktury `ProtoMessage`, a pomocí kanálu ji předá zpět TCP serveru, který tuto zprávu vezme, přidá k ní odesílatele, a skrz další kanál jí předá master serveru, který může zprávu na základě jejího obsahu a typu dále zpracovávat.

3.3.2 Reakce na zprávy

Přijatá a dekódovaná správa se pomocí kanálu posílá do takzvaného *master serveru*, který se stará o routing mimoherních zpráv a spouštění akcí s nich svázanými. Zároveň se stará o rozesílání výstupů z akcí na tyto požadavky.

Tyto akce jsou struktury splňující interface `Action`, díky kterému je možné všechny tyto akce uložit do dvouúrovňového slovníku. První úroveň používá jako klíč kontext hráče. Tím je už v podstatě návrhem zajištěno, že hráč, který není ve hře, nemůže poslat zprávu o pohybu. V druhé úrovni je pak klíčem celočíselný identivikátor zprávy.

Pokud tedy *master server* přijme zprávu, tak na základě kontextu odesílajícího hráče, a na základě typu zprávy, spustí danou akci. Tato akce vrátí *ActionResponse*, který kromě odpovědi dané akce obsahuje také kterým hráčům se má odpověď poslat. To z důvodu, aby bylo možné implementovat např. zprávu, která informuje hráče o startu hry. Jelikož požadavek posílá pouze jeden hráč, ale informaci o potvrzení startu musí dostat všichni hráčí v lobby.

3.3.3 Implementace hry

Samotnou hru v aplikaci řeší takzvaný *game server*. Tento *game server* je spouštěn pomocí zprávy *StartGame* ve vlastní gorutině. Po spuštění dojde k sestavení a inicializaci herního stromu, a následné spuštění samotného gameloopu. Tento gameloop se pak snaží 30× za sekundu vykonávat herní logiku. Implementace gameloopu umí řešit jak nedostatek, tak i přebytek výkonu, kdy se při nedostatku nečeká mezi jednotlivými cykly, čímž se server snaží „dohnat“ hru, a naopak při přebytku výkonu se čeká na správný čas, kdy se má provést další logika hry.

Herní logika je pak implementována tak, že každý uzel herního stromu má funkci `Process`, který dostává parametr `delta`, což je čas v sekundách, který uběhl od posledního spuštění, a pak parametr `playerMessages`, který obsahuje list příchozích zpráv od hráčů. Naslouchání na tyto zprávy si může každý node určit pomocí funkce `ListenMessages`, která umožňuje určit příchozí typy zpráv chce daný node naslouchat. Situace, kdy je více instancí jednoho nodu, ale daný hráč může ovládat jen některé, je vyřešená pomocí funkce `ListenMessages`, kde si daný node může profiltrovat reálné instance příchozích zpráv.

Přenos stavu hry k hráči je pak řešen zasíláním celého herního stromu. Tím se vytváří celkem vysoký datový tok, ale testy ukázaly, že vzhledem k povaze hry je toto řešení použitelné i v reálných podmínkách.

3.3.4 Herní objekty

V herním stromu jsou uloženy herní objekty typu `RootNode`, `Asteroid`, `Spaceship`, `Score` a `Projectile`.

RootNode

Tento node je v podstatě prázdný, a slouží jen a pouze jako node, který je v kořenu celého herního stromu, pod který se přidávají další herní objekty.

Asteroid

Asteroid je, jak již název hry napovídá, hlavním objektem celé hry. Asteroidy můžou mít různou velikost, s čímž souvisí i získané skóre za jeho zničení. Krom toho mají ještě svou pozici, vektor rychlosti a rotaci.

Klíč	Datový typ	Popis
pos_x	double	Pozice asteroidu na ose X
pos_y	double	Pozice asteroidu na ose Y
velocity_x	double	Složka X vektoru pohybu
velocity_y	double	Složka Y vektoru pohybu
rotation	double	Rotace asteroidu v radiánech
scale	double	Velikost asteroidu
value	integer	Získané skóre za zničení asteroidu

Tabulka 16: Specifikace veřejné části struktury `Asteroid`

Spaceship

Tento objekt reprezentuje konkrétní vesmírnou loď hráče, která má možnost střílet ostatní lodě a asteroidy, a také koliduje s asteroidy. Kolize s asteroidem způsobí smrt, přesunutí hráče na novou pozici, a udělí mu třísekundovou imunitu. Zároveň pokud hráč má imunitu, tak nemůže střílet.

Klíč	Datový typ	Popis
pos_x	double	Pozice lodi na ose X
pos_y	double	Pozice lodi na ose Y
velocity_x	double	Složka X vektoru pohybu
velocity_y	double	Složka Y vektoru pohybu
rotation	double	Rotace lodi v radiánech
player_name	string	Jméno hráče, kterému loď náleží
immune	boolean	Indikace, zda má hráč imunitu
reload_position	boolean	Vynucení aktualizace dat na straně klienta

Tabulka 17: Specifikace veřejné části struktury `Spaceship`

Score

`Score` je objekt, který je zodpovědný za řešení skóre jednotlivých hrářů ve hře. Jedná se o poměrně jednoduchý objekt, který s klientem sdílí pouze svou hodnotu, a jméno hráče, kterému náleží.

Klíč	Datový typ	Popis
score	integer	Hodnota skóre
player_name	string	Jméno hráče

Tabulka 18: Specifikace veřejné části struktury `Score`

Projectile

Tento node reprezentuje jednotlivé střely, které hráči vystřelují, a mají velmi podobné atribute, jako ostatní objekty vykreslitelné klientem.

Klíč	Datový typ	Popis
pos_x	double	Pozice střely na ose X
pos_y	double	Pozice střely na ose Y
velocity_x	double	Složka X vektoru pohybu
velocity_y	double	Složka Y vektoru pohybu
rotation	double	Rotace střely v radiánech

Tabulka 19: Specifikace veřejné části struktury `Projectile`

3.3.5 Použité kninhovny

Server používá knihovnu logrus na logování. Jazyk go sice má logovací knihovnu ve standardní knihovně, nicméně je poměrně strohá a ve spoustě mís-

tech nedostačující. Logrus nabízí především lepší manipulaci s formátem logu, jeho obarvováním v terminál, a nabízí i mnohem více log levelů.

Dále byla při vývoji použita knihovna go-spew, která umožňuje vypsání obsahu struktury v rámci všech zanořených strutkur. A to i za předpokladu, že jsou v dané struktuře ukazatele. Go-spew umí totiž tyto ukazatele automaticky dereferencovat, a vypsat jejich obsah, namísto obsahu ukazatele.

A nakonec byly využity knihovny yaml a jennifer. Yaml, protože v tomto formátu jsou definovány všechny zprávy serveru. A knihovna jennifer z důvodu schopnosti generování golang kódu. Tato metoda se v go používá jako náhrada chybějící podpory metaprogramování. Pomocí jennifer je konfigurační yaml soubor převeden na golang kód ve více souborech, což při vývoji ušetřilo spoustu práce, bez narušení struktury nebo abstrakce kódu.

3.4 Klient

Klient je implementováín v herním enginu godot. Obecně herní engine byl oproti standardním formulářovým frameworkům (Qt, Swing, JavaFX, ...) z důvodu povahu hry, která je realtime. Takováto hra by se dělala ve standardním formulářovém frameworku velmi obtížně. Ať už z důvodů technických omezení, tak z důvodu, že formulářové frameworky nejsou vůbec připravené pro tvorbu realtime hry. Velká část formulářového frameworku by tak zůstala nevyužita, jelikož by byla potřeba vlastní implementace většiny funkcionalit.

Konkrétně herní engine godot byl vybrán především z důvodu především dřívější zkušenosti, ale i z důvodu jeho jednoduchosti. Zároveň používá vlastní skriptovací jazyk GDScript (resp. GodotScript), který vychází z programovacího jazyka python. Nevychází z něj v plné míře. Bohužel chybí spousta věcí, na které jsou python programátoři zvyklí. Nicméně z hlediska syntaxe python velmi připomíná.

Vzhledem k tomu, že godot aktuálně nemá žádným způsobem vyřešené externí závislosti, a tudíž je používání externích knihoven celkem nepraktické, tak aplikace žádné nepoužívá.

3.4.1 Síť

Síť je v klientu řešena mnohem jednodušeji než na serveru. A to jednak z toho důvodu, že klient komunikuje vždy jen s jednou protistranou, tak i z důvodu, že godot není určen na to, aby si v něm programátor sám řešil síťovou komunikaci. API pro práci s TCP streamem je velmi omezené, a to dokonce tak, že práci se systémovými syscally hodnotím jako více přívětivou a mnohem pohodlnější. Godot má vlastní řešení multiplayeru, které funguje výborně, avšak ho vzhledem k povaze semestrální práce nebylo možné použít.

Síťová komunikace běží kompletně ve vlastním vlákně, a to především z důvodu výkonu. Jelikož je potřeba, aby v hlavním vlákně zbylo maximum výkonu pro herní logiku, aby byla zajištěna maximální plynulost samotné hry. Tato implementace podporuje znovupřipojení k serveru při jeho nedostupnosti, a zároveň je schopná pomocí událostí informovat hlavní logiku hry, o svém stavu. Při poslání zprávy má volající funkce možnost předat callback, a to jak na odpověď serveru, tak na timeout.

3.4.2 Herní menu

Herní menu je řešené pomocí vlastního malého ekosystému. Základem je zásobník, do kterého se ukládají instance jednotlivých menu, přičemž aktivní menu je to, která je na vrcholu zásobníku. To umožňuje pamatování si stavu při přechodu na předchozí menu, a díky tomu není potřeba znovu vyplňovat daný formulář, jelikož jsou v něm všechny uložené hodnoty. Zároveň tento ekosystém umožňuje buď všechna, nebo jen konkrétní menu vyresetovat, což umožňuje vymazání uživatelsky zadaného obsahu ve chvíli, kdy je to nelogické.

3.4.3 Hra

Samotná hra je pak řešena velmi jednoduše, a používá stejné herní objekty, které jsou již popsány (viz. Server). Používá také velmi jednoduché textury, volně dostupné fonty, a některé herní objekty mají grafické efekty. Zejména například herní loď vydává ze svého motoru částice, které se jinak chovají při neaktivitě, a jinak při pohybu. Díky těmto detailům hra nepůsobí tak prázdně, jak z důvodu své povahy ve skutečnosti je.

3.5 Sestavení a spuštění aplikace

Sestavení serveru je poměrně jednoduchá operace. Ohledně softwarového vybavění postačí jen nainstalovené go, a to ve verzi 1.13. Následně už stačí jen jít do složky serveru, a spustit příkaz `go build .`, případně `go run . IP_ADRESA PORT` pro kompilaci, a následné spuštění.

Sestavení klienta je podobně jednoduché. Postačí k tomu godot ve verzi 3.1 a jednoduché spuštění příkazu `godot -path cesta_ke_složce_s_klientem -export platforma výstupní_soubor`. Přičemž místo slova `platforma` se může nacházet `linux` nebo `windows`.

4 Závěr a zhodnocení práce

Závěrem bych chtěl říci, že semestrální práci považuji dle zadání za splňenou. Vzhledem ke zvoleným technologiím mi dala opravdu mnoho zkušeností, a mohl jsem si vyzkoušet technologie, ke kterým bych se jinak nejspíše nedostal.

Server je z mého hlediska zpracován poměrně kvalitně, zejména balík `net`, kterému bylo věnováno až nadměrné množství času, a prošel mnoha restrukturalizacemi a reimplementacemi. Jazyk go mi až na pár drobností vyhovoval, a určitě ho použiji i v budoucnu, pokud k tomu bude příležitost.

Co se týče klienta, tak si také myslím, že je zpracován adekvátně na to, že mám naprosto minimální zkušenosti s vývojem her. Co se týče enginu godot, tak nevím, zda bych ho zvolil i příště. Sám o sobě je to bezpochyby kvalitní engine, bohužel jsem se však potýkal s problémy týkající se GDScriptu. Ten vám „odpustí“ relativně velké množství chyb, a to až moc velké. Kolikrát se může stát, že daný skript spadne, a programátor se o tom ani nedozví. V lepším případě řekne, že nastal problém někde v útrobách enginu, nebo způsobí pád celé hry a vypsání call stacku. Tato skutečnost mě bohužel stála opravdu hodně času, a debugování samotné hry díky tomu bylo někdy velmi náročné. Až na vady spojené s GDScriptem mi engine jinak vyhovoval, nicméně příště bych určitě radši zkusil engine Unity3D.

Seznam tabulek

Seznam obrázků