

**POS-Tanks**

SEMESTRÁLNA PRÁCA

Vypracoval(a): **Filip Mikula, Patrik Vereš**

Študijná skupina: **5ZYI34**

Predmet: **Princípy operačných systémov**

Cvičiaci: *Ing. Patrik Rusnák, PhD.*

Obsah

[1. POS-Tanks 6](#_Toc124018661)

[2. Server 7](#_Toc124018662)

[2.1. Sockety 7](#_Toc124018663)

[2.2. Vlákna 7](#_Toc124018664)

[2.3. Synchronizačný problém 7](#_Toc124018665)

[2.4. Ďalšie problémy 8](#_Toc124018666)

[2.5. Štruktúra triedy Client 8](#_Toc124018667)

[2.5.1. Client(int id, std::string pName, float xPosition, float yPosition, DIRECTION direction, unsigned short port, sf::IpAddress ipAddress) 8](#_Toc124018668)

[2.5.2. ~Client() 8](#_Toc124018669)

[2.5.3. POSITION\* getPosition() 8](#_Toc124018670)

[2.5.4. CONNECTION\* getConnection() 8](#_Toc124018671)

[2.5.5. bool getFired() 8](#_Toc124018672)

[2.5.6. bool wasKilled() 8](#_Toc124018673)

[2.5.7. bool getLeft() 8](#_Toc124018674)

[2.5.8. bool wasScoreSent() 8](#_Toc124018675)

[2.5.9. int getClientId() 9](#_Toc124018676)

[2.5.10. int getKilledBy() 9](#_Toc124018677)

[2.5.11. int getScore() 9](#_Toc124018678)

[2.5.12. void updatePosition(float xPosition, float yPosition, DIRECTION direction) 9](#_Toc124018679)

[2.5.13. void setInitialPosition(float xPosition, float yPosition, DIRECTION direction) 9](#_Toc124018680)

[2.5.14. void setFired(bool fired) 9](#_Toc124018681)

[2.5.15. void resetPosition() 9](#_Toc124018682)

[2.5.16. void killed() 9](#_Toc124018683)

[2.5.17. void setLeft(bool left) 9](#_Toc124018684)

[2.5.18. void lockMutex() 9](#_Toc124018685)

[2.5.19. void unlockMutex() 9](#_Toc124018686)

[2.6. Štruktúra triedy Application 9](#_Toc124018687)

[2.6.1. Application() 9](#_Toc124018688)

[2.6.2. ~Application() 10](#_Toc124018689)

[2.6.3. void run() 10](#_Toc124018690)

[2.6.4. void initializeSocket() 10](#_Toc124018691)

[2.6.5. void waitForClients() 10](#_Toc124018692)

[2.6.6. void updateOfTanksPositions() 10](#_Toc124018693)

[2.6.7. void initializeGame() 10](#_Toc124018694)

[2.6.8. void sendData() 10](#_Toc124018695)

[2.6.9. void receiveData() 10](#_Toc124018696)

[2.7. Snímka obrazovky dokazujúca čistú pamäť 11](#_Toc124018697)

[3. Klient 12](#_Toc124018698)

[3.1. Sockety 12](#_Toc124018699)

[3.2. Vlákna 12](#_Toc124018700)

[3.3. Synchronizačný problém 12](#_Toc124018701)

[3.4. Ďalšie problémy 13](#_Toc124018702)

[3.5. Štruktúra triedy Application 13](#_Toc124018703)

[3.5.1. Application() 13](#_Toc124018704)

[3.5.2. ~Application() 13](#_Toc124018705)

[3.5.3. void run(sf::IpAddress ipAddress, int port, std::string playerName) 13](#_Toc124018706)

[3.5.4. int getPlayerScore() 13](#_Toc124018707)

[3.5.5. std::vector<Tank\*>\* getOtherTanks() 13](#_Toc124018708)

[3.5.6. void communicationWithServer() 14](#_Toc124018709)

[3.5.7. void connectToServer() 14](#_Toc124018710)

[3.5.8. void updatePositionsOfTanks() 14](#_Toc124018711)

[3.5.9. void waitForGameSettings() 14](#_Toc124018712)

[3.5.10. void render() 14](#_Toc124018713)

[3.5.11. void initializeWindow() 14](#_Toc124018714)

[3.5.12. void readClientInput() 14](#_Toc124018715)

[3.5.13. void checkBorders() 14](#_Toc124018716)

[3.5.14. void checkBulletCollision() 14](#_Toc124018717)

[3.5.15. void draw() 14](#_Toc124018718)

[3.5.16. void receiveData() 15](#_Toc124018719)

[3.5.17. void sendData() 15](#_Toc124018720)

[3.6. Štruktúra triedy Bullet 15](#_Toc124018721)

[3.6.1. Bullet() 15](#_Toc124018722)

[3.6.2. ~Bullet() 15](#_Toc124018723)

[3.6.3. void render(sf::RenderWindow& renderer) 15](#_Toc124018724)

[3.6.4. void shotBullet(float xPosition, float yPosition, DIRECTION direction) 15](#_Toc124018725)

[3.6.5. void setFired(bool fired) 15](#_Toc124018726)

[3.6.6. void setWasFiredAndSent() 15](#_Toc124018727)

[3.6.7. void resetWasFiredAndSent() 15](#_Toc124018728)

[3.6.8. sf::Vector2f getBulletPosition() 15](#_Toc124018729)

[3.6.9. sf::Vector2f getBulletSize() 15](#_Toc124018730)

[3.6.10. bool wasFired() 16](#_Toc124018731)

[3.6.11. bool wasFiredAndSent() 16](#_Toc124018732)

[3.6.12. bool checkBorders(std::vector<sf::RectangleShape\*>\* listOfWalls) 16](#_Toc124018733)

[3.6.13. void moveBullet() 16](#_Toc124018734)

[3.7. Štruktúra triedy Map 16](#_Toc124018735)

[3.7.1. Map() 16](#_Toc124018736)

[3.7.2. ~Map() 16](#_Toc124018737)

[3.7.3. std::vector<sf::RectangleShape\*>\* getListOfWalls() 16](#_Toc124018738)

[3.8. Štruktúra triedy Tank 16](#_Toc124018739)

[3.8.1. Tank() 16](#_Toc124018740)

[3.8.2. ~Tank() 16](#_Toc124018741)

[3.8.3. void moveUp(), void moveDown(), void moveLeft(), void moveRight() 16](#_Toc124018742)

[3.8.4. void fire() 16](#_Toc124018743)

[3.8.5. void rotate(DIRECTION dir) 17](#_Toc124018744)

[3.8.6. void render() 17](#_Toc124018745)

[3.8.7. void lockMutex() a void unlockMutex() 17](#_Toc124018746)

[3.8.8. void setDirection(DIRECTION dir) 17](#_Toc124018747)

[3.8.9. void setPlayerId(int pId) 17](#_Toc124018748)

[3.8.10. void setPlayerName(st::string name) 17](#_Toc124018749)

[3.8.11. void setLeft(bool left) 17](#_Toc124018750)

[3.8.12. void setScore(int score) 17](#_Toc124018751)

[3.8.13. sf::Sprite\* getSprite() 17](#_Toc124018752)

[3.8.14. int getDirection() 17](#_Toc124018753)

[3.8.15. int getPlayerId() const 17](#_Toc124018754)

[3.8.16. std::string getPlayerName() 17](#_Toc124018755)

[3.8.17. Bullet\* getBullet() 17](#_Toc124018756)

[3.8.18. bool getLeft() 17](#_Toc124018757)

[3.8.19. int getScore() 18](#_Toc124018758)

[3.9. Štruktúra triedy Menu 18](#_Toc124018759)

[3.9.1. Menu() 18](#_Toc124018760)

[3.9.2. ~Menu() 18](#_Toc124018761)

[3.9.3. void render() 18](#_Toc124018762)

[3.9.4. bool getAppClosed() 18](#_Toc124018763)

[3.9.5. sf::IpAddress getIpAddress() 18](#_Toc124018764)

[3.9.6. int getPort() 18](#_Toc124018765)

[3.9.7. std::string getName() 18](#_Toc124018766)

[3.9.8. void initializeWindow() 18](#_Toc124018767)

[3.9.9. void validate() 18](#_Toc124018768)

[3.10. Štruktúra triedy ShowScore 19](#_Toc124018769)

[3.10.1. ShowScore() 19](#_Toc124018770)

[3.10.2. ~Menu() 19](#_Toc124018771)

[3.10.3. void render() 19](#_Toc124018772)

[3.10.4. void initializeWindow() 19](#_Toc124018773)

[3.11. Snímka obrazovky dokazujúca čistú pamäť 20](#_Toc124018774)

[3.12. Príklad memory leak-u 20](#_Toc124018775)

# POS-Tanks

Semestrálna práca POS-Tanks sa skladá z dvoch častí. Z klientskej časti (grafické rozhranie, ovládanie tanku, komunikácia so serverom, ...) a zo serverovej časti (komunikácia s klientmi, riadenie hry, ...).

# Server

Po tom ako sa spustí serverová časť si aplikácia vyžiada vstup na počet hráčov, ktorí sa budú v danej hre vyskytovať. Následne sa čaká na pripojenie všetkých hráčov. Po tom ako sa pripojí hráč, hneď obratom dostane od servera jeho ID a jeho približné súradnice. Keď sa pripoja všetci hráči už len server rozpošle všetkým hráčom ich presné súradnice a dáta o ich súperoch.

Hra pokračuje až pokým sú v aréne pripojení hráči. Keď sa odpoja všetci hráči, server ukončuje svoju činnosť.

Po odpojení hráča mu server pošle aktuálnu štatistiku hry. To znamená že mu pošle jeho skóre a aktuálne skóre jeho protihráčov.

## Sockety

V serverovej časti aplikácie sú použité sockety na komunikáciu s klientom. Táto komunikácia prebieha oboma smermi. Komunikácia prebieha pomocou UDP protokolu.

UDP protokol sme zvolili z toho dôvodu, že ma najmenšiu réžiu, čiže je najrýchlejší a pri tomto počte v akom server komunikuje s klientom tolerujeme určité straty.

Socket používame z knižnice SFML. Je to z dôvodu, že ich sieťová knižnica je veľmi jednoduchá na správu a používanie. Zároveň z tejto knižnice používame aj Packet do ktorého vložíme dáta ktoré chceme poslať a následne keď druhá strana príjme tento paket tak jednoducho vie prečítať tieto dáta.

Server neustále počúva na zvolenom porte a keď príjme dáta od klienta tak následne pošle aktualizované dáta ostatným hráčom.

## Vlákna

Pri spustení servera a po počiatočnej inicializácií sa fungovanie rozdelí na 2 vlákna. Prvé vlákno slúži na počúvanie (čaká na správy od klienta). Druhé vlákno slúži na odosielanie dát. Zvolili sme tento prístup aby mohol nezávisle čakať na správy od klienta a zároveň aj odosielať aktuálne správy ostatným klientom.

## Synchronizačný problém

Hlavný synchronizačný problém bol taký, že dáta sa aktualizovali na jednom vlákne ale zároveň odosielali na druhom. Toto sme vyriešili pomocou Mutexu. Čiže po použití tohto prvku už nemohlo nastať, že by sa dáta čítali a zároveň prepisovali v jeden časový okamih.

## Ďalšie problémy

Pri serverovej časti aplikácie bolo ďalším problémom prísť na to ako často posielať aktualizované dáta klientom s tým, aby hra bežala plynulo ale zároveň aby sa nevyťažovala kapacita siete na 100%. Pretože keď sme pustili komunikáciu medzi klientom na rozličné vlákna bez žiadnej prestávky(sleep-u), tak nastalo extrémne veľké vyťaženie siete.

Takže sme zvolili prístup, že vlákno ktoré počúva na správy od klienta bude bežať neustále ale vlákno ktoré odosiela aktualizovane správy klientom bude fungovať iba potom ako príjme nejakú zmenu od klienta. Toto sme docielili pomocou mutexu a condition\_variable.

## Štruktúra triedy Client

* + 1. Client(int id, std::string pName, float xPosition, float yPosition, DIRECTION direction, unsigned short port, sf::IpAddress ipAddress)

Konštruktor triedy Client. Inštancia triedy Client predstavuje jedného hráča. Inicializuje počiatočne nastavenia hráča ako je jeho meno, ID, x a y pozícia, smer, IP adresu a port.

* + 1. ~Client()

Deštruktor triedy Client. Dealokuje všetky prostriedky a nastaví ich hodnotu na nullptr.

* + 1. POSITION\* getPosition()

Getter pre štruktúru pozícia. Týmto gettrom sprístupní klientove súradnice a jeho smer.

* + 1. CONNECTION\* getConnection()

Getter pre štruktúru pripojenia. Týmto gettrom sprístupní klientovu IP adresu a port.

* + 1. bool getFired()

Getter na informáciu či klient vystrelil. Vracia bool.

* + 1. bool wasKilled()

Getter na informáciu či bol klient zabitý. Vracia bool.

* + 1. bool getLeft()

Getter na informáciu či sa klient odpojil od hry. Vracia bool.

* + 1. bool wasScoreSent()

Getter na informáciu či bola hráčovi poslaná štatistika po odpojení. Vracia bool.

* + 1. int getClientId()

Getter na klientove ID. Vracia int.

* + 1. int getKilledBy()

Getter na ID hráča ktorý zabil tohto klienta. Vracia int.

* + 1. int getScore()

Getter na skóre hráča. Vracia int.

* + 1. void updatePosition(float xPosition, float yPosition, DIRECTION direction)

Funkcia na aktualizovanie aktuálnej polohy hráča. Aktualizuje štruktúru POSITION.

* + 1. void setInitialPosition(float xPosition, float yPosition, DIRECTION direction)

Funkcia na nastavenie počiatočnej polohy hráča ktorá sa neskôr používa na respawn po zabití.

* + 1. void setFired(bool fired)

Funkcia ktorá slúži na nastavenie či hráč vystrelil alebo nie.

* + 1. void resetPosition()

Funkcia ktorá slúži na resetovanie pozície hráča po zabití.

* + 1. void killed()

Funkcia ktorá slúži na nastavenie informácie, že hráč bol zabitý.

* + 1. void setLeft(bool left)

Funkcia ktorá slúži na nastavenie či hráč opustil hru.

* + 1. void lockMutex()

Funkcia slúži na uzamknutie mutexu.

* + 1. void unlockMutex()

Funkcia slúži na odomknutie mutexu.

## Štruktúra triedy Application

* + 1. Application()

Konštruktor triedy Application. Inicializuje potrebné polia a parametre.

* + 1. ~Application()

Deštruktor triedy Application. Dealokuje všetky inicializované atribúty.

* + 1. void run()

Funkcia ktorá spustí hru. Na začiatku sa inicializujú všetky potrebné sieťové prvky, následne sa počká na pripojenie hráčov, pošlú sa im nastavenia hry a v poslednom rade sa už iba rozdelí prevádzka na vlákna. Jedno pre počúvanie na správy od klientov a druhé na odosielanie správ klientom.

* + 1. void initializeSocket()

Funkcia ktorá slúži na inicializovanie socketu. To znamená že sa alokuje port na ktorom bude server počúvať.

* + 1. void waitForClients()

Funkcia ktorá si vyžiada vstup na koľko klientov má čakať (koľko bude hráčov) a následne čaká na ich pripojenie, kde im hneď aj pošle ich ID a približné súradnice kde sa majú postaviť.

* + 1. void updateOfTanksPositions()

Funkcia ktorá slúži na presné nastavenie počiatočnej pozície hráčov. Klient pošle serveru presné súradnice, ktoré sú prepočítané na základe od veľkosti tanku.

* + 1. void initializeGame()

Funkcia ktorá slúži na poslednú časť inicializácie hry. Server rozpošle všetkým klientom informácie o ich súperoch a to aké je ich ID, meno, pozícia a smer.

* + 1. void sendData()

Funkcia vlákna, ktorá slúži na odosielanie dát klientom.

* + 1. void receiveData()

Funkcia vlákna, ktorá slúži na prijímanie dát od klienta.

## Snímka obrazovky dokazujúca čistú pamäť

Obrázok, na ktorom je text, snímka obrazovky, počítač

Automaticky generovaný popis

# Klient

Po spustení klientskej časti aplikácie sa používateľovi zobrazí okno hlavného menu, v ktorom si používateľ zadá prezývku, IP adresu a port servera, na ktorý sa chce pripojiť. Pokiaľ nevyplní požadované údaje nie je možné hru spustiť.

Po spustení hry sa klient pripojí k serveru, od ktorého získa približné súradnice tanku, jeho smer, ID a počet hráčov, ktorí budú spolu hrať. Klient si tieto súradnice upraví v závislosti na veľkosti tanku a taktiež si nastaví farbu tanku podľa obdržaného ID. Následne klient odošle upravené súradnice späť na server. Ďalším krokom je čakanie na príchod správy od servera, v ktorej sú informácie o ostatných hráčoch. Po jej obdržaní si klient tieto informácie uloží do vektora Tankov. Keď už máme všetky inicializačné procesy za sebou, tak sa vytvoria tri vlákna (pre odosielanie, prijímanie správ a pre grafiku).

Hra beží pokiaľ ju hráč neukončí. Keď hráč ukončí hru pošle sa serveru správa, že ju ukončil. Aplikácia počká na správu s aktuálnym skóre všetkých hráčov. Nakoniec zobrazí hráčovi jeho skóre a skóre ostatných hráčov.

## Sockety

Rovnako ako pri serverovej časti sme použili knižnicu SFML, pomocou ktorej pracujeme so Socket-om a Packet-om. Klient komunikuje so serverom pomocou UDP protokolu. Správy, ktoré posielame a prijímame máme rozčlenené na viacero vymenovaných typov, podľa čoho určujeme aký typ informácie odosielame a prijímame.

## Vlákna

Po spustení klienta a potrebnej inicializácií pred začiatkom hry sa beh aplikácie rozdelí na 3 vedľajšie vlákna. Jedno vlákno slúži na prijímanie správ od servera. Čaká na správy, ktoré server posiela a následne ich spracováva. Ďalšie vlákno slúži na odosielanie správ serveru. Posledné vlákno sa stará o grafickú časť aplikácie. Zobrazuje okno s hrou, kontroluje vstupy od používateľa, okraj mapy, zásahy a vykresľuje tanky a guľky na okno.

## Synchronizačný problém

Hlavným synchronizačným problémom bolo rozloženie komunikácie klient-server na dve vlákna a pridanie mutexu, aby sme mohli simultánne posielať aj prijímať správy a zároveň nenastal problém s prístupom k dátam, ktoré by sme v jednom vlákne čítali a v druhom ich menili.

Ďalej sme riešili synchronizáciu vystrelenej guľky tak, že odosielame iba správu o jej vystrelení v danom okamžiku spolu so súradnicami tanku. Túto informáciu rozpošle server ďalším klientom a tí si ju už spracujú.

## Ďalšie problémy

Medzi ďalšie riešené problémy patrí implementácia knižníc. Pri prvom návrhu hry sme využívali knižnicu SDL (tu sme si vyskúšali pridanie obdĺžniku, ktorý mohol strieľať a pohybovať sa po mape). Pri tejto knižnici bolo potrebné správne umiestniť .dll súbory do systémových priečinkov a správne nastaviť CMake (prechod na MakeFile sme vykonali neskôr už aj s novou knižnicou). Po chvíli sme sa rozhodli prejsť na knižnicu SFML, z ktorej sme popri grafickej časti použili aj sieťovú pre komunikáciu so serverom. Pri tejto knižnici sme sa zbavili nutnosti pridávať .dll súbory do systémových priečinkov.

Ďalším problémom bol fakt že klient sa spúšťa na Windowse a nie je tam natívna podpora pre Makefile. Tu bolo treba stiahnuť MinGW balíček v ktorom sa nachádza aj príkaz make. Po správnom nalinkovaní knižníc pri Makefili nastal ďalší problém a to taký že staršie a 32-bitové balíčky MinGW nepodporujú vlákna, mutexy a condition\_variables. Čiže Makefile nám nechcel skompilovať náš kód. To sa napokon vyriešilo tím že sme našli a nainštalovali 64-bitovú MinGW knižnicu v ktorej sa nachádza aj podpora pre Possixove vlákna.

V neposlednom rade bolo ďalším problémom kontrola memory leakov. Na serveri s tým nebol problém pretože na Linuxe existovala podpora pre Valgrind. Lenže na Windowse sme takúto možnosť nemali, preto sme museli prekopať celý projekt do Visual Studia (hlavne linkovanie grafickej knižnice) a tam sme použili štruktúru na kontrolovanie memory leakov ktorú sme používali aj na Údajových štruktúrach.

## Štruktúra triedy Application

* + 1. Application()

Konštruktor triedy Application. Inicializuje počiatočne atribúty ako napr.: font, vektor tankov a počiatočné nastavenia textu, ktorý bude reprezentovať používateľove meno.

* + 1. ~Application()

Deštruktor triedy Application. Dealokuje všetky prostriedky a nastaví ich hodnotu na nullptr.

* + 1. void run(sf::IpAddress ipAddress, int port, std::string playerName)

Funkcia, ktorá spusti hru. Najskôr sa nastavia potrebné atribúty, potom sa spusti funkcia communicationWithServer() a nakoniec sa spustia vlákna pre komunikácia a grafiku.

* + 1. int getPlayerScore()

Getter na skóre hráča. Vracia int.

* + 1. std::vector<Tank\*>\* getOtherTanks()

Getter na vektor tankov, do ktorého sa ukladajú tanky ostatných hráčov.

* + 1. void communicationWithServer()

Slúži na vytvorenie pripojenia k serveru, získanie súradnic, ID, atď. Následne upraví súradnice vygenerované serverom a odošle mu ich späť. Príjme nastavenia hry a informácie o protihráčoch od servera a spracuje ich.

* + 1. void connectToServer()

Pošle prázdnu správu serveru, na ktorú server odpovie pridelením ID a pozície tanku.

* + 1. void updatePositionsOfTanks()

Pošle už upravené súradnice tanku (v závislosti na jeho veľkosti) serveru.

* + 1. void waitForGameSettings()

Príjme správu, ktorá obsahuje informácie o polohe všetkých protihráčov a vytvorí reprezentáciu ich tankov, ktoré si uloží do vektora.

* + 1. void render()

Funkcia, ktorá sa stará o spustenia a správne ukončenie okna. Taktiež obsahuje cyklus, ktorý kontroluje vstupy od hráča, kolízie a vykresľuje tanky a guľky na okno.

* + 1. void initializeWindow()

Vytvorí okno, nastaví mu výšku a šírku, štýl, obnovovaciu frekvenciu a nastaví ho aktívnym.

* + 1. void readClientInput()

Zaznamenáva vstupy od klienta ako stláčanie kláves pre pohyb tanku, výstrel a zatvorenie okna. Taktiež zaznamenáva zatvorenie okna pomocou tlačidla na zatvorenie okna.

* + 1. void checkBorders()

Táto funkcia zabezpečuje aby tank nevyšiel mimo hracej plochy alebo neprechádzal cez steny.

* + 1. void checkBulletCollision()

Táto funkcia rieši zásah guľky do tanku protivníka. Ak sme nejakého zasiahli zapíše jeho ID do atribútu idOfKilledPlayer.

* + 1. void draw()

Vykresľuje na okno tank hráča a jeho prezývku a taktiež všetky tanky protihráčov a guľky.

* + 1. void receiveData()

Slúži na prijímanie správ od servera. Po prijatí správy ju spracuje na základe typu správy. V prípade správy STATUS aktualizuje pozície tankov protihráčov, prípadne výstrel nejakého z nich. Ak ide o správu KILLED “respawn-e“ hráča daného IDčkom. Pri správe END uloží skóre všetkých hráčov. Pri poslednej správe PLAYER\_QUIT si poznačí, ktorý hráč ukončil hru.

* + 1. void sendData()

Odosiela správy serveru. Rovnako rozlišuje ich typ a obsah.

## Štruktúra triedy Bullet

* + 1. Bullet()

Konštruktor triedy Bullet. Inicializuje atribúty: speed, diameter, štvorec, ktorý reprezentuje guľku a či bola vystrelená a či boli o informácie o nej poslané.

* + 1. ~Bullet()

Deštruktor triedy Application. Dealokuje štvorec guľky a nastaví jeho hodnotu na nullptr.

* + 1. void render(sf::RenderWindow& renderer)

Vykresli guľku na okne.

* + 1. void shotBullet(float xPosition, float yPosition, DIRECTION direction)

Nastaví pozíciu guľky na pozíciu dela tanku.

* + 1. void setFired(bool fired)

Nastaví, že guľka bola vystrelená.

* + 1. void setWasFiredAndSent()

Nastaví, že informácie o guľke boli poslané.

* + 1. void resetWasFiredAndSent()

Nastaví fireAndSent\_ na false.

* + 1. sf::Vector2f getBulletPosition()

Vracia pozíciu guľky.

* + 1. sf::Vector2f getBulletSize()

Vracia veľkosť guľky.

* + 1. bool wasFired()

Vracia informáciu, či bola guľka vystrelená.

* + 1. bool wasFiredAndSent()

Vracia informáciu, či boli poslané informácie o guľke.

* + 1. bool checkBorders(std::vector<sf::RectangleShape\*>\* listOfWalls)

Vracia informáciu, či guľka nevyšla mimo mapu alebo nenarazila do steny.

* + 1. void moveBullet()

Posunie guľku v jej smere o jej rýchlosť.

## Štruktúra triedy Map

* + 1. Map()

Konštruktor triedy Map. Inicializuje všetky steny a pridáva ich do vektoru stien.

* + 1. ~Map()

Deštruktor triedy Tank. Dealokuje všetky steny a vektor a nastaví jeho hodnotu na nullptr.

* + 1. std::vector<sf::RectangleShape\*>\* getListOfWalls()

Getter, ktorý vracia vektor všetkých stien na mape.

## Štruktúra triedy Tank

* + 1. Tank()

Konštruktor triedy Tank. Inicializuje počiatočne atribúty ako napr.: či hráč opustil hru, textúru tanku, jeho rýchlosť a strelu.

* + 1. ~Tank()

Deštruktor triedy Tank. Dealokuje všetky prostriedky a nastaví ich hodnotu na nullptr.

* + 1. void moveUp(), void moveDown(), void moveLeft(), void moveRight()

Funkcia, ktorá otočí tank daným smerom a posunie ho o jeho rýchlosť.

* + 1. void fire()

Vystrelí guľku. Kontroluje, či už má tank prebité.

* + 1. void rotate(DIRECTION dir)

Otočí tank smerom podľa parametra.

* + 1. void render()

Vykresli tank na okno a ak vystrelil, tak vykresli aj jeho guľku.

* + 1. void lockMutex() a void unlockMutex()

Slúžia na zamknutie a odomknutie mutexu.

* + 1. void setDirection(DIRECTION dir)

Nastaví smer tanku podľa parametra.

* + 1. void setPlayerId(int pId)

Nastaví atribút playerId\_ podľa parametra.

* + 1. void setPlayerName(st::string name)

Nastaví prezývku hráča.

* + 1. void setLeft(bool left)

Nastavuje atribút, ktorý hovorí o tom, či hráč opustil hru.

* + 1. void setScore(int score)

Zapíše do atribútu skóre hráča.

* + 1. sf::Sprite\* getSprite()

Vracia objekt, ktorý graficky reprezentuje tank.

* + 1. int getDirection()

Getter pre smer tanku. Vracia int.

* + 1. int getPlayerId() const

Getter, ktorý vracia ID hráča.

* + 1. std::string getPlayerName()

Getter, ktorý vracia meno hráča.

* + 1. Bullet\* getBullet()

Getter, ktorý vracia objekt guľky.

* + 1. bool getLeft()

Getter, ktorý vracia, či hráč opustil hru.

* + 1. int getScore()

Getter na skóre hráča.

## Štruktúra triedy Menu

* + 1. Menu()

Konštruktor triedy Menu. Inicializuje všetky potrebné atribúty a ich vlastnosti.

* + 1. ~Menu()

Deštruktor triedy Menu. Dealokuje všetky prostriedky a nastaví ich hodnotu na nullptr.

* + 1. void render()

Vykresľuje okno a kontroluje vstupy hráča. Hráč sa môže prepínať medzi jednotlivými poľami, písať do nich, spustiť hru alebo zavrieť aplikáciu.

* + 1. bool getAppClosed()

Getter, ktorý vracia, či bola aplikácia zatvorená.

* + 1. sf::IpAddress getIpAddress()

Getter, ktorý vracia IP adresu, ktorú hráč zadal do poľa.

* + 1. int getPort()

Getter, ktorý vracia port, ktorý hráč zadal do poľa.

* + 1. std::string getName()

Getter, ktorý vracia hráčovu prezývku, ktorú zadal do konkrétneho poľa.

* + 1. void initializeWindow()

Vytvorí okno, nastaví mu výšku a šírku, štýl, obnovovaciu frekvenciu a nastaví ho aktívnym.

* + 1. void validate()

Kontroluje, či hráč môže spustiť hru. Musí mať zadanú ľubovoľnú prezývku. IP adresa a port sa kontrolujú pomocou regulárnych výrazov.

## Štruktúra triedy ShowScore

* + 1. ShowScore()

Konštruktor triedy ShowScore. Inicializuje všetky potrebné atribúty a ich vlastnosti.

* + 1. ~Menu()

Deštruktor triedy ShowScore. Dealokuje okno a nastaví jeho hodnotu na nullptr.

* + 1. void render()

Vykresľuje okno a kontroluje vstupy hráča (či hráč nezavrel dané okno). V okne sa zobrazí skóre hráča a jeho protihráčov.

* + 1. void initializeWindow()

Vytvorí okno, nastaví mu výšku a šírku, štýl, obnovovaciu frekvenciu a nastaví ho aktívnym.

## Snímka obrazovky dokazujúca čistú pamäť

## Príklad memory leak-u