Dokumentacja

# 1. Działanie gry

Gra sieciowa, maksymalnie 4 graczy. Czas trwania rundy: 5 min. Wygrywa gracz który zabije największą ilość graczy. ID gracza który wygrał pokazywane jest po zakończeniu rundy przez 10 sek. Po tym czasie pozycja graczy restartuje się i gra zaczyna się od nowa.

Można strzelić pociskiem co 2 sek. Liczba pocisków jest nieograniczona.

Gracz którym aktualnie sterujemy ma kolor zielony, natomiast pozostałe czołgi mają kolor czerwony.

Sprawdzanie czy pocisk trafił na przeszkode lub inny czołg odbywa się na serwerze który w odpowiednim momencie wysyła informacje do klienta, żeby usunąć pocisk z listy pocisków.

# 2. Dokumentacja

## A. Klient

### Display

Klasa odpowiadająca za wyświetlanie grafiki.

* *Qlist* **playerList** – lista graczy.
* *Qlist* **missileList** – lista pocisków.
* *Qlist* **scoreboard** – tablica wyników.
* *Qlist* **map** – lista obiektów tworzących mape.
* *Glint* **mapWidth, mapHeight** – wymiary mapy, szerokość i wysokość wyrażona w pikselach.
* *Gldouble* **ammoProgress** – postęp ładowania pocisków 0 – 100%
* *bool* **drawScore** – zmienna logiczna odpowiadająca za sprawdzanie czy potrzeba rysować tablice wyników
* *int* **gameTimer –** czas gry
* *bool* **gameStatus –** status gry: **true –** gra w trakcie, **false –** koniec rundy, wyświetla id zwycięzcy
* *int* **playerID –** ID gracza
* *void* **drawHudBar(), drawScoreboard(), drawTimer(), drawResults()** - metody odpowiadające za rysowanie poszczególnych elementów.

### **MainWindow**

* *(slot) void* **infoReceived()**  - slot odpowiadający za odbieranie wiadomości z serwera
* *(slot) void* **onTimer()** - slot odpowiadający za wykonywanie czynności co określony czas
* *Qtimer* **timer –** główny timer
* *int* **roundTimer –** timer odpowiadający za mierzenie czasu rundy
* *bool* **roundTimerEnabled –** wartość logiczna odpowiadająca za włączenie lub wyłączenie timera rundy
* *int* **mt –** timer dla sprawdzenia możliwości strzału
* *bool* **mtON –** wartość logiczna odpowiadająca za włączenie lub wyłączenie timera strzału
* *int* **playerID –** id gracza
* *int* **timerInterval –** okres timera
* *bool* **keyUp, keyDown, keyRight, keyLeft, keyQ, keyE, keySpace –** wartość logiczna sprawdzająca czy dany klawisz jest wciśnięty
* *QtcpSocket* **socket –** socket odpowiadający za komunikacje z serwerem
* *void* **connectBox()** - metoda odpowiadająca za wyświetlanie okienko do łączenia się z serwerem
* *void* **movePlayer()** - metoda odpowiadająca za poruszanie gracza oraz wykonywanie strzału
* *void* **updateGame(QString &data) –** metoda odpowiadająca za uaktualnianie stanu gry po odbiorze informacji z serwera

### Tank

* *Void* **print()** - metoda służąca do wyświetlania czołgu
* *void* **setRotation, setCannonRotation, setPos, setColor, setCannonColor –** settery ustawiające odpowiednie wartości
* *void* **rotate(), rotateCannon()** - metoda obracająca czołg lub wieżyczke czołgu
* *Glfloat* **getXPos(), getYPos(), getRotation(), getCannonRotation()** - gettery zwracające odpowiednie wartości
* *void* **move()** - porusza czołgiem
* *bool* **canShoot()** - zwraca **true** jeśli conajmniej minęły 2 sek. od poprzedniego strzału lub **false** jeśli nie minęły
* *void* **canShoot(*bool value*) –** ustawia wartość atrybutu **ableToShoot**
* *bool* **canMove(*GLfloat speed, QList<Tank \*> \*tanksList, QList<Obstacle \*> \*map, GLfloat top, GLfloat right, GLfloat bottom, GLfloat left, GLfloat offset = 0*) –** sprawdza czy czołg może się poruszyć. Sprawdza kolizje między innymi czołgami, przeszkodami oraz granicami map.
* *Int* **id –** id czołgu
* *Glfloat* **xPos, yPos –** współrzędne położenia czołgu
* *bool* **ableToShoot –** wartość logiczna mówiąca czy czołg może strzelić
* *Glfloat* **rotation, cannonRotation –** rotacja czołgu lub wieżyczki
* *Glfloat* **color[4], cannonColor[4] –** tablica kolorów, odpowiednio – czerwony, zielony, niebieski, kanał alpha

### Missile

* *Void* **setXPos(*Glfloat value*), setYPos(*Glfloat value*) –** settery wartości pozycji pocisku
* *void* **setAngle(*Glfloat value*) –** setter dla kątu poruszania się pocisku
* *void* **print()** - rysuje pocisk
* *void* **move()** - metoda poruszająca pociskiem
* *Glfloat* **getXPos(), getYPos()** - gettery wartości położenia pocisku
* *GLfloat* **xPos, yPos –** atrybuty określające połozenie pocisku
* *GLfloat* **color[4] –** tablica kolorów pocisku
* *GLfloat* **angle –** kąt poruszania się pocisku
* *GLfloat* **speed –** prędkośc poruszania się pocisku

### Score

* *void* **setDeaths**(***int*** ***value***) –*setter atrybutu* ***deaths***
* *void* **setKills**(***int*** ***value***) – setter atrybutu **kills**
* *void* **reset**() - resetuje wartość atrybutów **kills** oraz **deaths**
* *int* **getKills**() - getter atrybutu **kills**
* *int* **getDeaths**() - getter atrybutu **deaths**
* *QString* **toString**() - zwraca *Qstring* z **id** gracza oraz wartościami zabić oraz śmierci
* *int* **tankID –** id czołgu
* *int* **kills –** liczba zabić
* *int* **deaths –** liczba śmierci

### Obstacle

* *void* **print()** - wyświetla przeszkode
* *GLfloat* **getX()** - getter atrybutu **x**
* *GLfloat* **getY()** - getter atrybutu **y**
* *GLfloat* **getRad()** - getter atrybutu **rad**
* *GLfloat* **x, y –** położenie środka przeszkody
* *GLfloat* **rad –** promień przeszkody

## **B. Serwer**

### **MainWindow**

* *QList*<*Tank* \*> **playersList –** lista graczy
* *QList*<*Missile* \*> **missileList –** lista pocisków
* *QList*<*Score* \*> **scoreboard –** tablica wyników
* *QList*<*Obstacle* \*> **map –** lista przeszkód skladających się na planszę
* *double* **defaultPosTab[4][3] –** tablica domyślnych pozycji dla graczy od 1 - 4
* *void* **setDefaultPos()** - metoda ustalająca domyślne wartości w tablicy **defaultPosTab**
* *void* **setupMap()** - tworzy planszę
* *void* **updateGame(QString &data) –** przekształca podany Qstring w wartości do aktualizacji gry
* *void* **resetGame()** - restartuje pozycje graczy oraz wyniki w tablicy wyników
* *QTcpServer* **\*server –** wskaźnik dla serwera
* *QNetworkSession* **\*networkSession –** aktualna sesja
* *QList*<*QTcpSocket* \*> **clients –** lista połączonych klientów
* *void* **onConnectMessage(QTcpSocket \*\_socket) –** wysyła wiadomość do nowego gracza oraz pozostałych informująca o dołączeniu nowego gracz
* *QString* **getPlayerInfo(int id) –** zwraca Qstring o kliencie z danym **id** przygotowany do wysłania do klienta
* *QString* **getMissileInfo(int id) –** zwaraca Qstring o pocisku o danym **id** przygotowany do wysłania do klienta
* *QString* **getScoreInfo(int id, bool shorten = false) –** Zwraca Qstring o wyniku gracza o danym **id**
* *QTimer* **gameTimer –** główny timer gry
* *int* **timerInterval –** okres timera
* *int* **roundTimer –** timer rundy
* *bool* **roundTimerEnabled –** określa czy timer rundy jest włączony
* *bool* **roundEnd –** określa czy dana runda jest ukończona
* *int* **roundEndTimer –** timer odliczający 10 sek. pomiędzy rundami
* *void* **openSession()** - tworzy nową sesję
* *void* **onTimer()** - slot wykonywany przez główny timer
* *void* **newConnection()** - dodaje nowego klienta do listy socketów
* *void* **clientDisconnect()** - rozłącza klienta oraz wysyła informacje do pozostałych graczy
* *void* **newInfo()** - odbiera informacje od socketów i przekazuje ją do metody **updateGame()**

### Tank

* *void* **setRotation(double value) –** setter dla atrybutu **rotation**
* *double* **getRotation()** - zwraca wartość atrybutu **rotation**
* *void* **setPos(double x, double y) –** ustawia pozycje gracza
* *double* **getXPos()** - zwraca wartośc atrybutu **yPos**
* *double* **getYPos()** - zwraca wartośc atrybutu **xPos**
* *double* **getCannonRotation()** - zwraca wartość atrybutu **cannonRotation**
* *void* **setCannonRotation(double value)** - setter dla atrybutu **cannonRotation**
* *int* **id –** id gracza
* *double* **xPos, yPos –** pozycja czołgu
* *double* **rotation, cannonRotation –** rotacja czołgu/wieżyczki

### Missile

* *void* **setXPos(double value) –** setter atrybutu **xPos**
* *void* **setYPos(double value)** - setter atrybutu **yPos**
* *double* **getXPos()** - getter atrybutu **xPos**
* *double* **getYPos()** - getter atrybutu **yPos**
* *double* **getDirection()** - getter atrybutu **angle**
* *int* **tankID –** id czołgu który wystrzelił pocisk
* *void* **move()** - metoda poruszająca pociskiem
* *int* **hit(QList<Tank\*> \*tanksList) –** zwraca **id** trafionego czołgu lub **-1** jeśli nie trafi nikogo
* *bool* **canMove(double top, double right, double bottom, double left, QList<Obstacle \*> \*map) –** zwraca **true** jeśli pocisk nie trafił na ściane lub przeszkode
* *double* **speed –** prędkość poruszania się pocisku
* *double* **xPos, yPos –** wartości położenia pocisku
* *double* **angle –** kąt ruchu pocisku

### **Score**

* *void* **addKill()** - dodaje 1 do atrybutu **kills**
* *void* **addDeath()** - dodaje 1 do atrybutu **deaths**
* *int* **getKills()** - zwraca wartość atrybutu **kills**
* *int* **getDeaths()** - zwraca wartość atrybutu **deaths**
* *void* **reset()** - restartuje wartości atrybutów **kills, deaths**
* *int* **kills –** liczba zabić
* *int* **deaths –** liczba śmierci

### Obstacle

* *QString* **getData()** - zwraca Qstring przygotowany do wysłania do klienta
* *double* **getX()** - zwraca wartość atrybutu **x**
* *double* **getY()** - zwraca wartość atrybutu **y**
* *double* **getRad()** - zwraca wartośc atrybutu **rad**
* *double* **x, y, rad –** wartości położenia przeszkody oraz jej promienia