Documentation for Concurrency Project
Group 8

Irfan Ali
Siddharth Goel
Arnav Kumar

Details About the Game

This is a 2 player shooting game with a moving target. Both players should try to hit the target and score as many points in the limited time frame. The player who scores the maximum points wins. One of the players plays as server and the other plays as client.

Starting the Game

First player to start the game hosts the server. The next player to join that server becomes the client in the game. No more connections can be established now.

Explaining the Code:

Different Objects in the Game

The one using graphics are: Players, bows, arrows, target, scoreboard and score.
The one not using graphics are: Gamestate, server and client, and screenupdate.

Explaining the Classes(Briefly)

myplayer1 : This class is used for creating players. It inherits from QGraphicsPixmapItem

bow : This class is used for creating bows to be used by players to shoot arrows. It inherits from QGraphicsPixmapItem.

arrow : This class is used for creating arrows to be used by players to hit the target. It inherits from QObject and QGraphicsPixmapItem.

target : This class is used to create the target object to be hit by players. It inherits from QObject and QGraphicsPixmapItem.

score and scoreboard : These classes are used to display Player's and Opponent's score.

gamestate : This class stores the state of the game at each instance.

server : This class is used to host a server. It inherits from QObject.

client : This class is used to join a server. It inherits from QObject.

screenupdate : This class is used to update the positions of the game objects on both server and client screen. It updates the values from the gamestate.

Functioning of the game

Explaining Gamestate Class : This class stores state of the game at each instant. The state of the game is defined using :
int id; // 0 if server, 1 if client
QGraphicsScene *startscene; // scene in which game is running
QPointF Player1Position; // stores position of your player
QPointF Player2Position; // stores position of opponent player
QPointF TargetPosition; // stores position of the target
bool isArrow1; // tells if player1's arrow is in the scene
bool isArrow2; // tells if players2's arrow is in the scene
QPointF Arrow1Position; //Stores position of player1's arrow

QPointF Arrow2Position; //Stores position of player2's arrow
qreal Arrow1Angle; //Stores the angle of arrow of player1
qreal Arrow2Angle; //Stores the angle of arrow of player2
QPointF Arrow2Position; //Stores the position of player2's arrow
qreal Bow1Angle; //Stores the angle of bow of player1
qreal Bow2Angle; //Stores the angle of bow of player2
int hit; // 1 if client hits the target else 0
int points1; //Stores player1's score
int points2; //Stores player2's score
QJsonObject getJsonObject(); // JSonObject is used to pass message

The player is given the choice to either host as server or join as a client to an existing server.

Playing as Server : The server initializes the objects, Players, target and scoreboard and adds them to the scene. When the server is started the screenupdate updates the screen every 50ms.

What happens in server : The server has a QWebsocketServer object which listens at a given ip and port number. When the startserver function is called the server starts listening and it connects the signal "newConnection" to the slot function connection setup in class Server. If the client is ready then the connection is made otherwise if already the server is playing then the client is asked to come later. Once the connection is set up, gameloop parameters are changed and the gamestate is passed by server to client and client to server to update their respective screens. Gamestate is sent using JSon object and JSon Document which are processed by processBinary function and process text function.

What happens in client : The client initializes server connection and starts the game as soon connection is set up. ProcessBinary processes the

gamestate sent by server and updates using screenupdate. Gamestate is sent using JSon object and JSon Document which are processed by binary function and process text function.

Screenupdate : A timer is running in main which updates the gamestate processed by processBinary in client and server after every 50ms.

Scoring : Gamestate contains the score parameters (points1, points 2). These are updated by screenUpdate. Whenever arrow collides with the bug we increase the score of respective players' score.

How does arrow hit the target : If the position of the arrow is on the target, we increase the score of the respective player, remove the arrow from the scene and then reset the position of the target. If the client hits the target it need to update an integer value hit(initialised to 0) to 1. This sends a signal to server to know that the client has hit the target and then reset the position of target because only the server can update the target.To ensure two arrows don't hit the target at the same time, the arrow acquires a mutex lock before hitting the target and then releases the lock only after updating the target and position of arrow. The players' whose arrow acquires the lock gets the chance to score. Where ever the target's position is being accessed mutex lock is used to prevent data inconsistency.

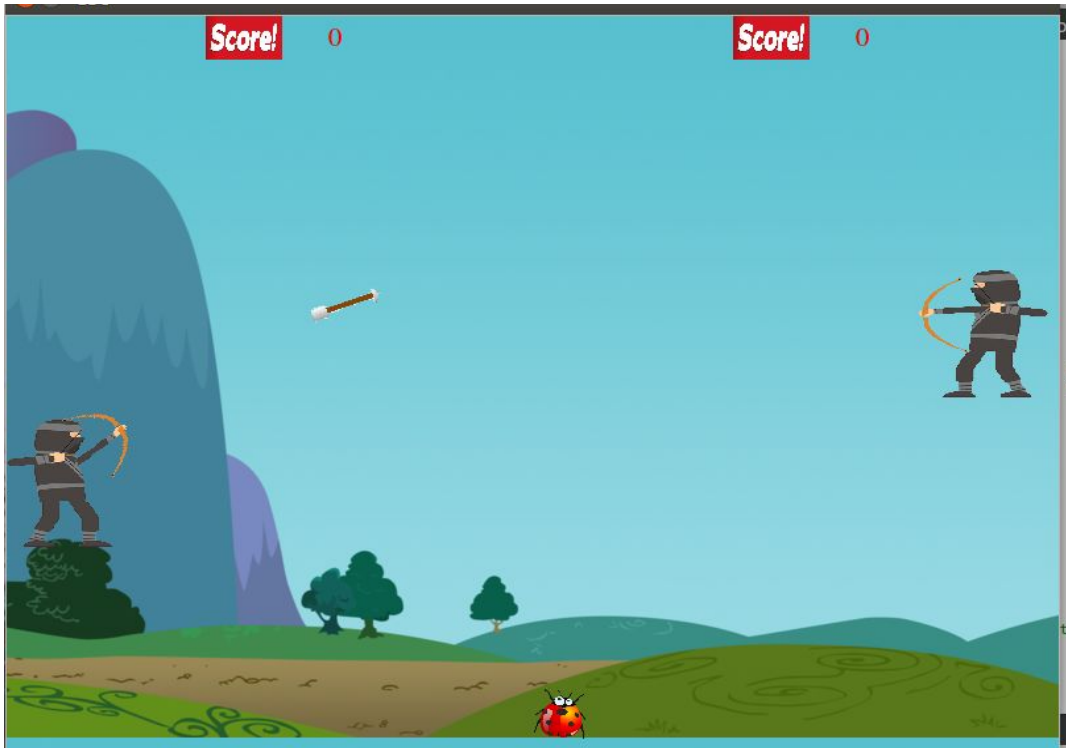
Usage of Concurrency

The position of target can be simultaneously changed by two or more functions which are being run by using signals and Qtimer.
So to prevent data inconsistency or data race or deadlocks, we have used a Qmutex lock.
The lock is declared as global variable. Whenever any function tries to access or change the target attributes, it has to first acquire the mutex lock and then do the computations. After the computations are done, the process has to release (unlock) the mutex lock. This prevents different

process from accessing target attributes at the same time. Hence, achieving concurrency.

Resource Folder contains the .png files of different graphics objects.

GamePlay




Score!