TYPE SHOOTER
PROJECT DOCUMENTATION

KHETHOKUHLE KHUZWAYO

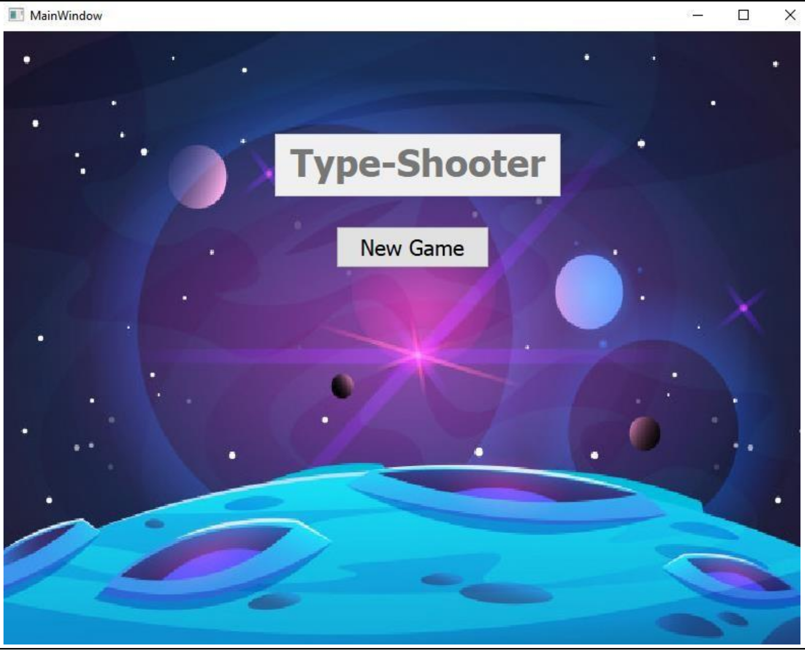
Contents

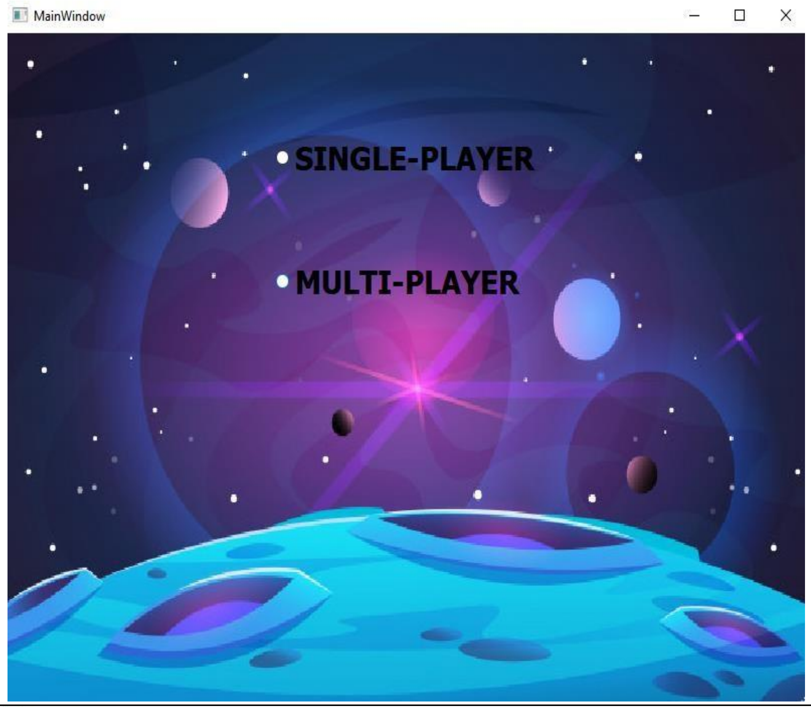
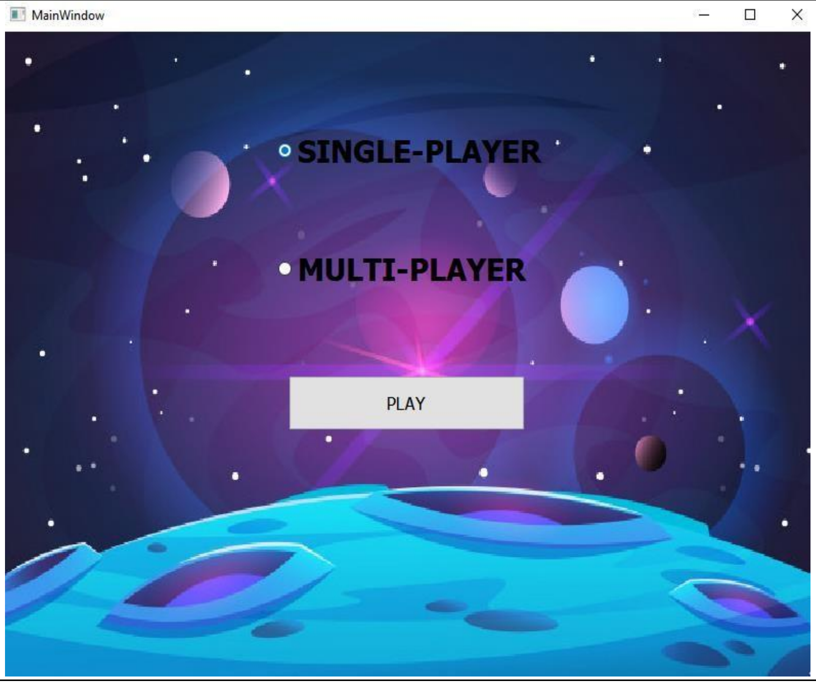
Introduction

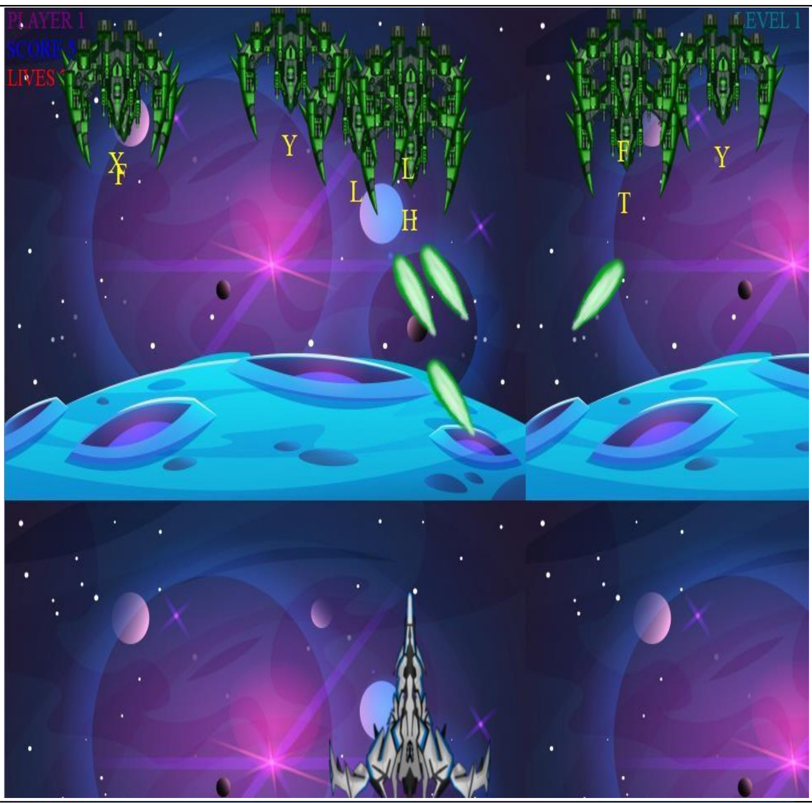
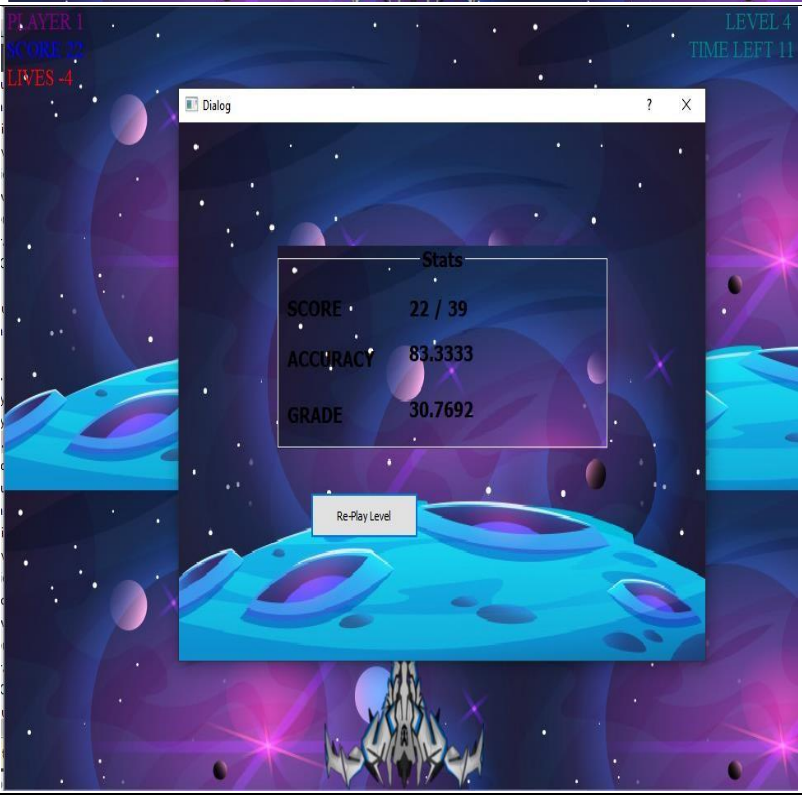
The name of the educational game created is Type-Shooter. The aim was to instill and/or improve a user's keyboarding skills. Its goal is to teach children the three fundamentals of keyboard typing, which are correct hand/finger placement on keyboard, increased typing speed, and typing accuracy. These three fundamentals were at the center of the game, together with the inclusion of graphic visuals to ensure that learning occurs while the gamer has fun. This game is aimed at primary schoolers in grades 3 to 7, as the human mind is easier to mould when the person is still young. The game is score based and the user earns a point with each correct letter he/she is able to type in that appears on the screen, moving downwards towards the player's character. With each correct letter a bullet will launch and destroy the corresponding letter, whilst each incorrect letter results in lower weighted average. The user loses a point with each letter that descends below the screen without the user typing it in. There are four levels of progression each with increased difficulty than the previous one as with each increasing level, the number of letters spawned onto the screen increases as well as the speed at which they descend. Each level lasts for 60 seconds and the user progress to the next if they managed to meet the minimum average criteria or outlast the time without losing too many points (subject to the difficulty of the level). This game will be presented to a school called Sherwood Primary School.

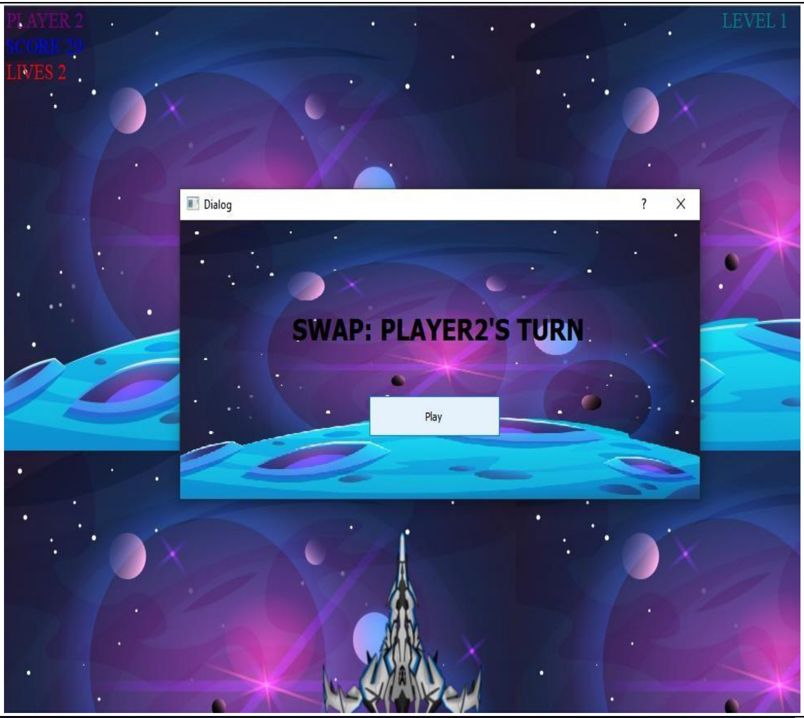
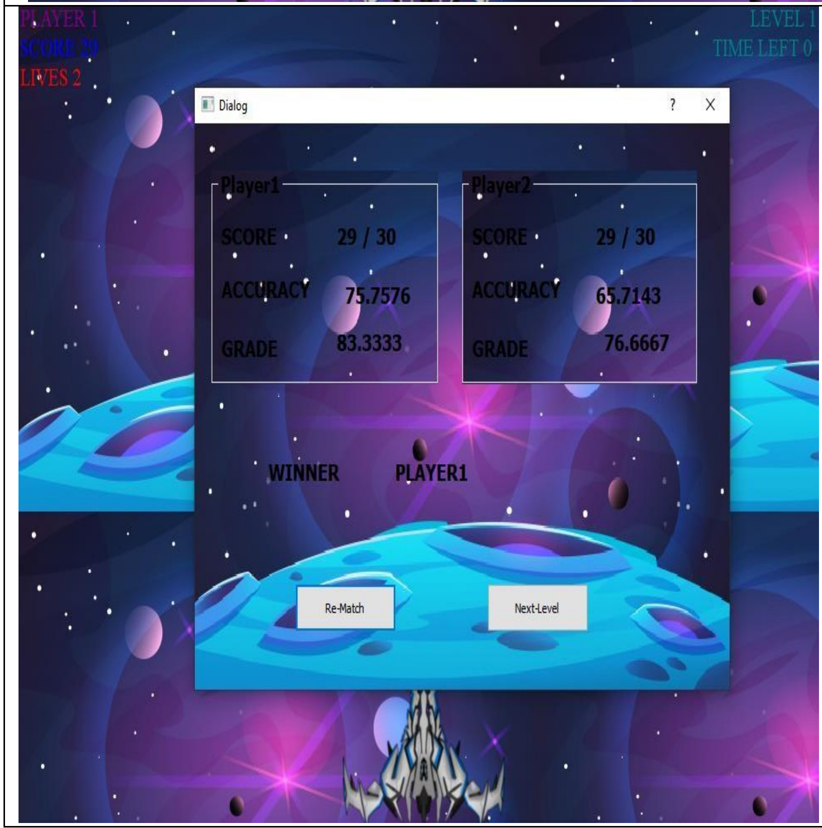
…

Screenshots

Screenshot	Explanation
	The Image is of the starting window upon running the application. The user is to select the "New Game" button to move on to the next window.

	Upon selecting the “New Game” button, the user will be presented with this screen and select which mode that they want to play in, either alone (SinglePlayer) or against a friend (Multi-Player).
	Then, immediately after selecting one of the radio buttons, the user will have the button “PLAY” appear on the screen which will take the user to the actual game.

 PLAYER 1 SCORE 5 LIVES LEVEL 1 X F Y L L H F Y T	Once the "PLAY" button has been pushed, the fun begins! The image on the left displays the game in action. The user will have 60 seconds to type as many letters as he can. Each time he does this, a bullet will appear to destroy that the corresponding enemy. With each enemy destroyed the score increases and if an enemy escapes the lives will be decreased. Once the number of lives decreases to zero, the level is lost.
 PLAYER 1 SCORE 22 LIVES -4 LEVEL 4 TIME LEFT 11 Dialog Stats SCORE 22 / 39 ACCURACY 83.3333 GRADE 30.7692 	Here is a screen displaying the user's performance in this level of difficulty. This screen is instantiated immediately after the level and the score as well as the grade of the user is displayed on the screen. The significance of this screen will be discussed below. Here the user did not progress therefore cannot go through to the next level so can only replay this one.

 PLAYER 2 SCORE 29 LIVES 2 LEVEL 1 Dialog SWAP: PLAYER2'S TURN 	If the user is playing in multi-player mode, player one will play first then after he is finished, this message will display on screen indicating to the users that it is now player 2' turn.
 PLAYER 1 SCORE 29 LIVES 2 LEVEL 1 TIME LEFT 0 Dialog Player1 SCORE 29 / 30 ACCURACY 75.7576 GRADE 83.3333 Player2 SCORE 29 / 30 ACCURACY 65.7143 GRADE 76.6667 WINNER PLAYER1 	In Multi-player mode, after player2 has finished playing the users are then both given their stats. This screen will appear showing how they both faired off at this level of difficulty and against each other. Their stats are then compared against each other and a winner is declared. Here as we can see both users managed destroy the same number of enemies but their grades are not the same as player 1 was less prone to making mistakes than player2, hence him being the winner. Users are given an option to Re-play or move to the next level

Real and Educative Game

The game is a game

It is intrinsically motivating in that the player will be required to shoot out as many letter enemies as they can in the time given, whilst trying to ensure that he/she does not run out of lives first. The game is level based, meaning that the user progresses to more challenging stages ensuring to keep the user captivated. It has rules in that at the end of the round, the player will be scored, and based on whether the player meets the minimum criteria or not, he/she may progress to the next phase of the game. The criteria is that the number of lives the user has at a given level must not be 0 at the end, as this will signify that the user is fairly competent in this level and can progress to the next. The user loses a life every time an enemy is able to escape without the user successfully shooting it i.e. by typing it's character. The player may replay a level to improve his/her score should they see the need or progress to the next round (if they qualify too). It is interactive and graphical as the user must type in the name of the alien which is a single character from the alphanumeric keyboard. The name will appear on the alien's body, and so as soon as the alien appears, the player will be able to shoot at it. It has a score in that the user will be scored on how fast he/she can type in a character as it appears, and how accurate he/she is as there will be a negative scoring for typing in characters that do not appear on the screen.

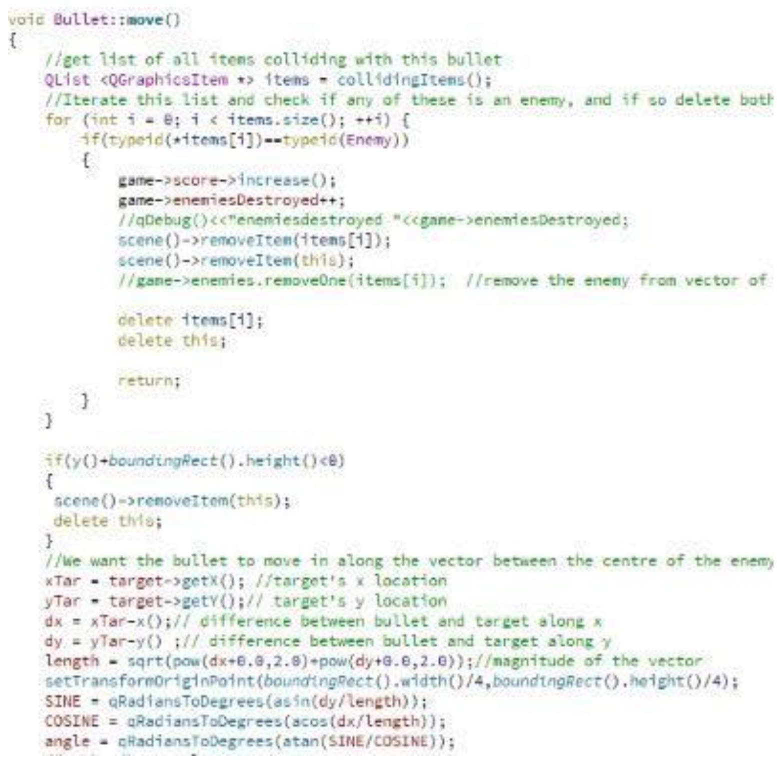
The game is educative

It has fantasy elements and that is seen by the presence of aliens that the player is required to shoot out. Fantasy fosters children's cognitive and language skills. It provides meaningful feedback at the end of the game which means that the player will know where to improve (speed or precision). This feedback details the players Characters Per Minute ratio as how many characters typed in correctly within the round. It also includes the player's typing accuracy as a percentage of how many times the user correctly types in a letter to how many times the user types. The user is then graded according to an adjusted score that considers all the times the player typed in whether it's correctly or incorrectly, and the number of characters the user did not type in (this is the number of lives lost). The less characters missed and incorrectly typed in, the more proficient the user is at this level and the higher the user's grade. Also, each level has a maximum time limit of one minute, with each level having more characters required to type in than the previous one. So, a grade of 100% at level one for example, implies that the user is very proficient at typing at 30 characters per minute (roughly 6 words per minute), and a score of 100% at level 4 implies proficiency at about 120 characters per minute (roughly 30 words per minute). So, we see that as the user gradually progresses through the levels of this game, the user's Character Per Minute typing ratio increases along with them and so the game is educative in that regard

Programming Techniques

1. Function

<u>**Screenshot:**</u>



```
void Bullet::move()
{
    //get list of all items colliding with this bullet
    QList <QGraphicsItem *> items = collidingItems();
    //Iterate this list and check if any of these is an enemy, and if so delete both
    for (int i = 0; i < items.size(); ++i) {
        if(typeid(*items[i])==typeid(Enemy))
        {
            game->score->increase();
            game->enemiesDestroyed++;
            //qDebug()<<"enemiesdestroyed "<<game->enemiesDestroyed;
            scene()->removeItem(items[i]);
            scene()->removeItem(this);
            //game->enemies.removeOne(items[i]);  //remove the enemy from vector of

            delete items[i];
            delete this;

            return;
        }
    }

    if(y()+boundingRect().height()<0)
    {
     scene()->removeItem(this);
     delete this;
    }
    //We want the bullet to move in along the vector between the centre of the enemy
    xTar = target->getX(); //target's x location
    yTar = target->getY();// target's y location
    dx = xTar-x();// difference between bullet and target along x
    dy = yTar-y() ;// difference between bullet and target along y
    length = sqrt(pow(dx+0.0,2.0)+pow(dy+0.0,2.0));//magnitude of the vector
    setTransformOriginPoint(boundingRect().width()/4,boundingRect().height()/4);
    SINE = qRadiansToDegrees(asin(dy/length));
    COSINE = qRadiansToDegrees(acos(dx/length));
    angle = qRadiansToDegrees(atan(SINE/COSINE));
```

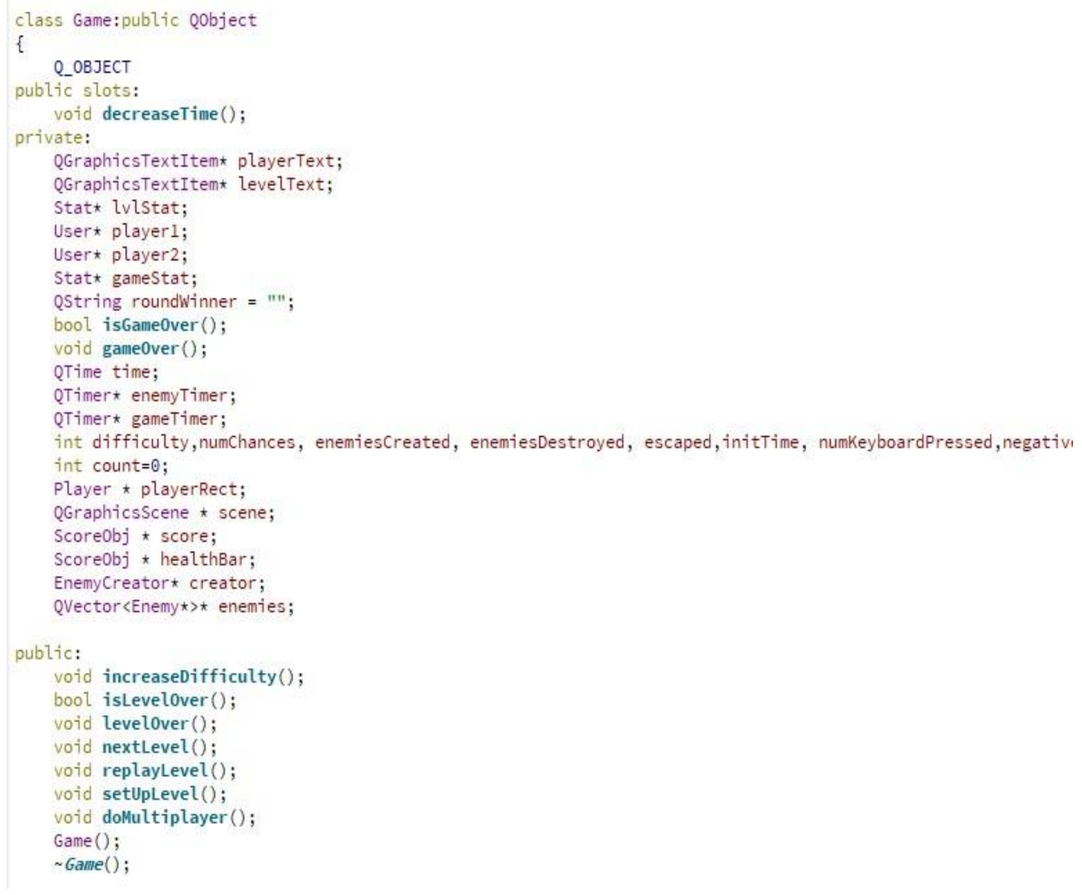
<u>Motivation:</u>

The function "move" is a member function of the Bullet class that will move the bullet instance on the screen towards the designated target letter. It takes the current location of a targeted enemy(letter) and computes the unit vector between the player character and the designated target letter and then move along that direction towards that target letter. The letters themselves are also not stationery, meaning that with each new move the bullet makes, it needs to re-compute the unit vector it is moving along. So, it becomes apparent that if we were to hardcode this, the program would be un-necessarily bloated with repetitive code each time we required the bullet to move hence why a function was necessary here.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	Function objective is to replace repetitive lines of code and I believe this function achieves that

2. Class

Screenshot: Game.h



```
class Game:public QObject
{
    Q_OBJECT
public slots:
    void decreaseTime();
private:
    QGraphicsTextItem* playerText;
    QGraphicsTextItem* levelText;
    Stat* lvlStat;
    User* player1;
    User* player2;
    Stat* gameStat;
    QString roundWinner = "";
    bool isGameOver();
    void gameOver();
    QTime time;
    QTimer* enemyTimer;
    QTimer* gameTimer;
    int difficulty,numChances, enemiesCreated, enemiesDestroyed, escaped,initTime, numKeyboardPressed,negativ
    int count=0;
    Player * playerRect;
    QGraphicsScene * scene;
    ScoreObj * score;
    ScoreObj * healthBar;
    EnemyCreator* creator;
    QVector<Enemy*>* enemies;

public:
    void increaseDifficulty();
    bool isLevelOver();
    void levelOver();
    void nextLevel();
    void replayLevel();
    void setUpLevel();
    void doMultiplayer();
    Game();
    ~Game();
```

Motivation:

For any large complex software project, classes are fundamental as they break down the big task at hand into smaller sub-tasks. This game is no different as it consists of classes such as "Player", "Enemies", "MainWindow, and many more. Now all these knew nothing about the next class, they only handle what is required of them so it became apparent that they will need to communicate somehow in order to produce the desired system and so the class called "Game" was created. This class takes all the defined classes and data types as attributes and defines behaviours as member functions for these attributes according to the requirements of the game. Now this allows for any client that is calling this software to not have to deal with every single class defined but just call this one instead and this provides a layer of abstraction between the client and the module which then allows for more flexibility, portability and re-usability hence why I defined this class.

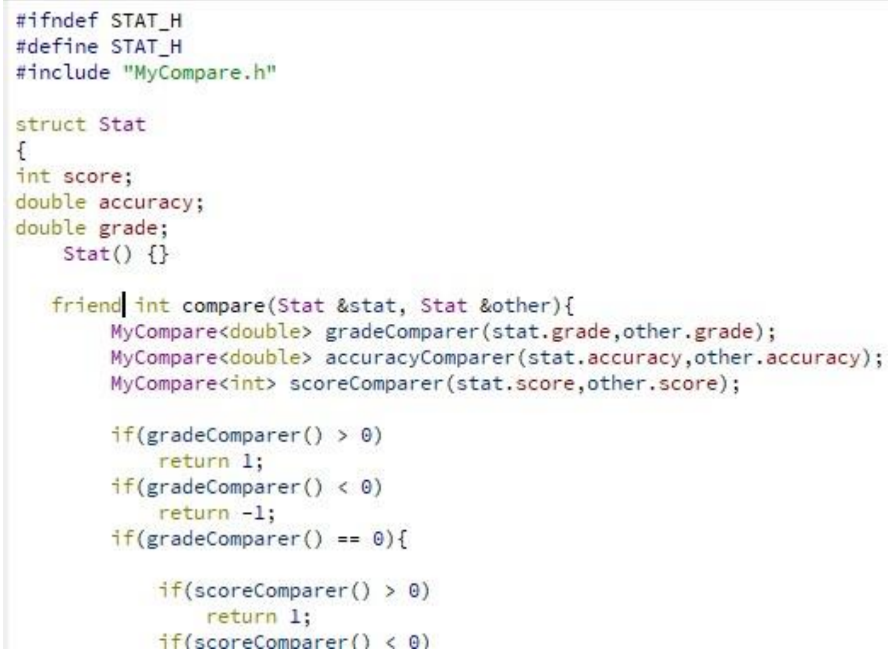
How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		

Completely	X	The objective of a class is to provide user defined types or entities together with the it’s attributes and behaviourisms (member function) to modify and or change it’s state to give the functionality desired by

		the user. As we can see, the "Game" entity has the ability to modify its state (e.g setUpLevel(), decreseTime()) which provide the much needed game functionality

3. Struct

Screenshot:



```
#ifndef STAT_H
#define STAT_H
#include "MyCompare.h"

struct Stat
{
int score;
double accuracy;
double grade;
    Stat() {}

   friend int compare(Stat &stat, Stat &other){
        MyCompare<double> gradeComparer(stat.grade,other.grade);
        MyCompare<double> accuracyComparer(stat.accuracy,other.accuracy);
        MyCompare<int> scoreComparer(stat.score,other.score);

        if(gradeComparer() > 0)
            return 1;
        if(gradeComparer() < 0)
            return -1;
        if(gradeComparer() == 0){

            if(scoreComparer() > 0)
                return 1;
            if(scoreComparer() < 0)
```

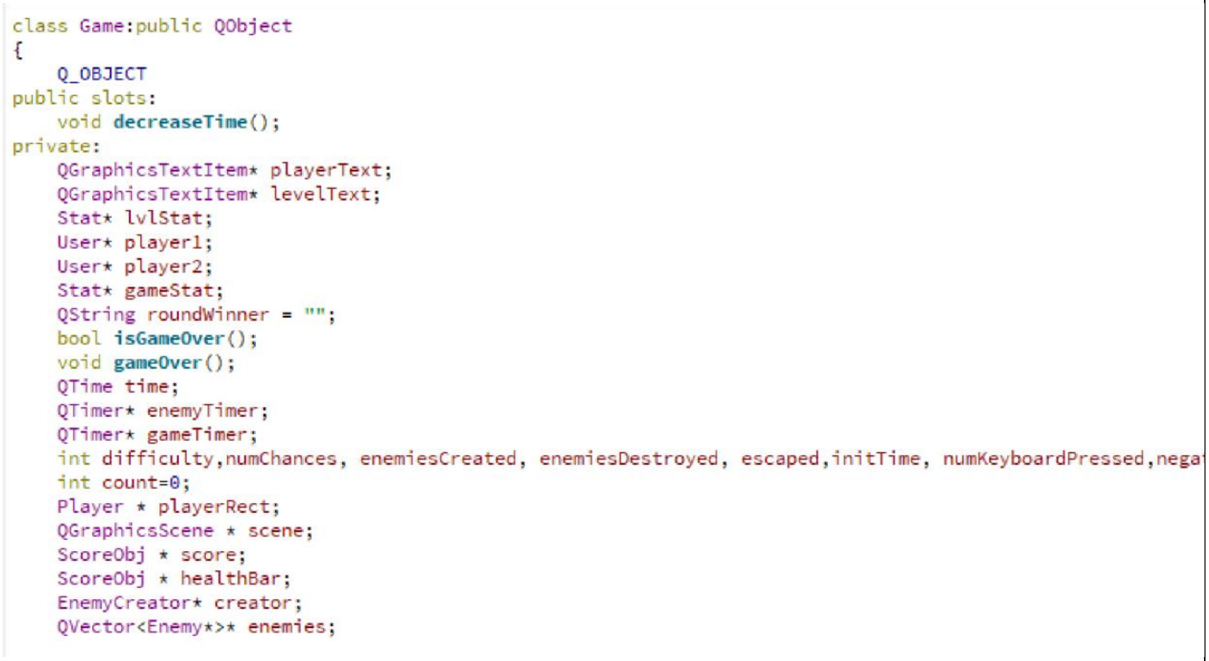
Motivation:
I opted to use a structure here as a I needed to define a data-type that will store a user's statistics, but not be modified after creation as after each round of the game a user is given feedback according to how many characters per minute they were able to type, how accurate they were and what is their final grade for that round. Seeing that there is no need for modifying, this data type didn't need member functions and so this is why I opted for it. The function is a friend of this struct which is used to compare variables of this struct, therefore it not a member function.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		

Completely	X	The objective of a Struct is to provide user defined variables that will not be modified after
		instantiation, and that is precisely how it is used.

4. Pointer

Screenshot: Pointer Attributes of Game.h



```
class Game:public QObject
{
    Q_OBJECT
public slots:
    void decreaseTime();
private:
    QGraphicsTextItem* playerText;
    QGraphicsTextItem* levelText;
    Stat* lvlStat;
    User* player1;
    User* player2;
    Stat* gameStat;
    QString roundWinner = "";
    bool isGameOver();
    void gameOver();
    QTime time;
    QTimer* enemyTimer;
    QTimer* gameTimer;
    int difficulty,numChances, enemiesCreated, enemiesDestroyed, escaped,initTime, numKeyboardPressed,nega
    int count=0;
    Player * playerRect;
    QGraphicsScene * scene;
    ScoreObj * score;
    ScoreObj * healthBar;
    EnemyCreator* creator;
    QVector<Enemy*>* enemies;
```

Motivation:

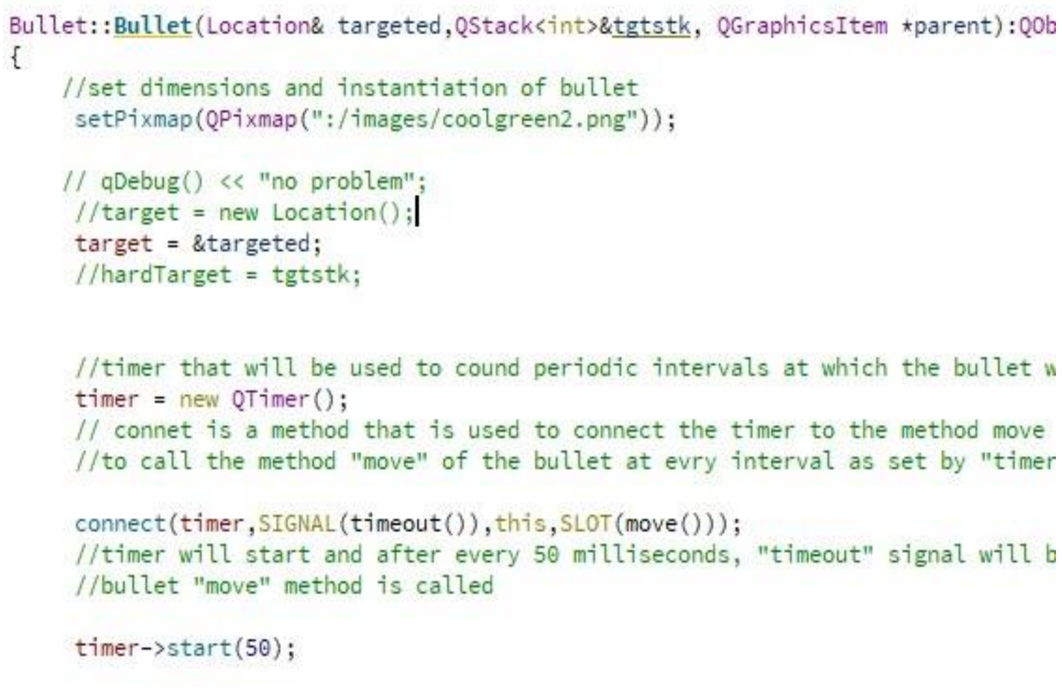
The screenshot above is from the class Game.h , which was discussed above. Seeing that this is such a large project, there were many new types that had to be defined by the user to provide the needed functionality to the game. Now many of these types had to become attributes of this class, and thus were attributes which required to be allocated and de-allocated run-time memory. Pointers provide a more robust way of doing this, as opposed to class instances. Since they only contain the address of a variable and not its content, they are less memory demanding which results in reduced space and time complexity of the program. All this was more than enough motivation to use pointers in a large project such as this one.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		

Completely	X	Pointers Objective is to provide the memory location of variables. For user defined variables(and also primitive variables), they can allocate and de-allocate runtime memory to them which is precisely how they are used in this program (mostly) thus satisfying the objective of the pointer.

5. Reference

Screenshot:



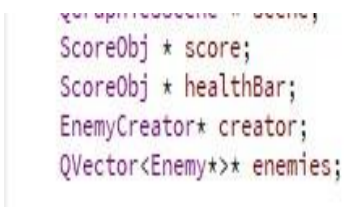
```
Bullet::Bullet(Location& targeted,QStack<int>&tgtstk, QGraphicsItem *parent):QOb
{
    //set dimensions and instantiation of bullet
     setPixmap(QPixmap(":/images/coolgreen2.png"));

    // qDebug() << "no problem";
     //target = new Location();
     target = &targeted;
     //hardTarget = tgtstk;


     //timer that will be used to cound periodic intervals at which the bullet w
     timer = new QTimer();
     // connet is a method that is used to connect the timer to the method move
     //to call the method "move" of the bullet at evry interval as set by "timer

     connect(timer,SIGNAL(timeout()),this,SLOT(move()));
     //timer will start and after every 50 milliseconds, "timeout" signal will b
     //bullet "move" method is called

     timer->start(50);
```

Motivation:
The above screenshot is the constructor of the “Bullet” class. The variable “targeted” is a reference parameter which instantiates the Bullet attribute “target”, which is used by the function member function (of Bullet) “move” to compute the required trajectory as discussed above and so it was crucial for the data received by the constructor to be accurate because remember the target is not stationery, meaning that the target’s current location may not be the same location when the bullet is required to move the solution was to declare a Location type pointer that will point directly to the actual “targeted” variable in memory rather than create as a local copy which will go out of scope once the constructor finishes executing. This was only possible by making “targeted” a reference parameter.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	Objective of reference is to pass the memory address of a
		variable only if we want the function to directly affect the contents of this address which is exactly how it used here as it used to retrieve the exact location of the enemy as it moves on screen hence we have met this objective

6. Vector

Screenshot: Declaration of Qvector(Vectors in Qt) Pointer



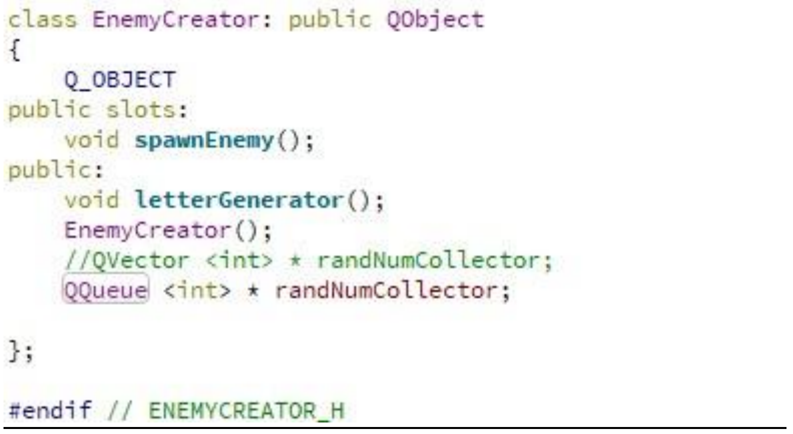
```
[illegible]
ScoreObj * score;
ScoreObj * healthBar;
EnemyCreator* creator;
QVector<Enemy*>* enemies;
```

Motivation:
The above screenshot is the vector which stores the enemies(letters) which are currently on screen. The reasoning for storing these in this data structure is that it is not known how many enemies at a given time will appear on the screen so size must be variable, upon the user typing in the correct letter a bullet will be instantiated at destroy that enemy(letter). When this occurs, that enemy needs to be removed from the data structure, but it's may be the first, last or somewhere in the middle and so any data structure which is strict with the entry and removal of elements cannot be used here , and so a Vector seemed to be the most appropriate for the task at hand.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	Vector Objective is to store objects of the same type with constant time random access to all of them. This achieved in how the vector is used to store characters which are on the screen as it is not known which character the user will type in first and so access to all of them should be more or less the same time complexity.

7. Data Structure

Screenshot:



```
class EnemyCreator: public QObject
{
    Q_OBJECT
public slots:
    void spawnEnemy();
public:
    void letterGenerator();
    EnemyCreator();
    //QVector <int> * randNumCollector;
    QQueue <int> * randNumCollector;

};

#endif // ENEMYCREATOR_H
```

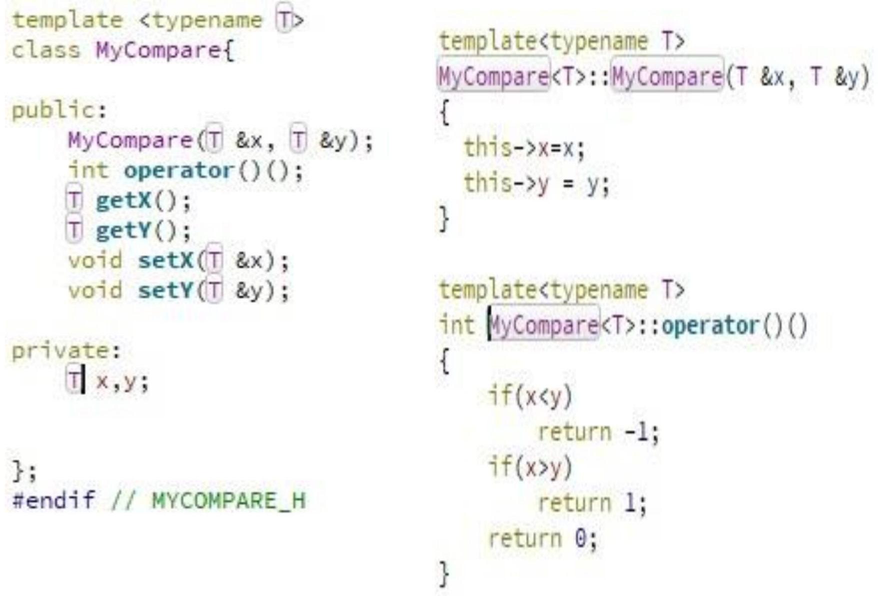
Motivation:

The above screenshot is Queue data structure specialized to store int variables. It is an attribute of the enemy creator class, which is responsible for the creation of enemies on the screen. The data structure is used to store random numbers ranging from 65-90 (asci codes of capital letters). At any given time, the letters appearing on the screen should not ideally be the same, so these numbers are stored in a data structure and are later used to generate letters to appear on the screen. Once a number has been used, it we remove it from the data structure and so it became apparent that these numbers enter and exit the data structure in a First-in-first-out fashion and so it was more logical to store them in a Queue.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	Queue objective is to store objects of the same type in a first in first out manner. This is exactly how it was used in our program so the Queue objective was met

8. Class Template

Screenshot: MyCompare.h(left) and MyCompare.cpp(right).



```
template <typename T>
class MyCompare{

public:
    MyCompare(T &x, T &y);
    int operator()();
    T getX();
    T getY();
    void setX(T &x);
    void setY(T &y);

private:
    T x,y;


};
#endif // MYCOMPARE_H
```

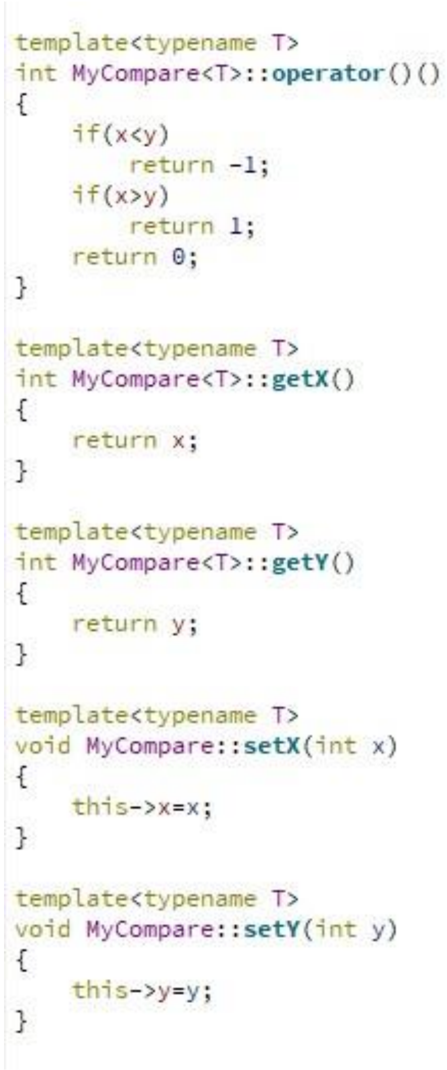
```
template<typename T>
MyCompare<T>::MyCompare(T &x, T &y)
{
  this->x=x;
  this->y = y;
}

template<typename T>
int MyCompare<T>::operator()()
{
    if(x<y)
        return -1;
    if(x>y)
        return 1;
    return 0;
}
```

Motivation:
In Multiplayer mode, if two users are competing against each other, a winner would have to be established somehow. The only way to this is to compare their respective stats and so their stats have to be comparable, but now the Stat data type is a Structure so operator overloading was not an option hence I used a function object class that takes template arguments and compares them by their natural ordering and returns the result.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		
Completely	X	The objective of a class template is to provide a template entity which will be initialized to a concrete type at compile time. A function object being a class makes it an entity, and the fact that it takes template arguments in the constructor make it a template class satisfying the objective

9. Function Template

Screenshot: MyCompare.cpp



```
template<typename T>
int MyCompare<T>::operator()()
{
    if(x<y)
        return -1;
    if(x>y)
        return 1;
    return 0;
}

template<typename T>
int MyCompare<T>::getX()
{
    return x;
}

template<typename T>
int MyCompare<T>::getY()
{
    return y;
}

template<typename T>
void MyCompare::setX(int x)
{
    this->x=x;
}

template<typename T>
void MyCompare::setY(int y)
{
    this->y=y;
}
```

Motivation:

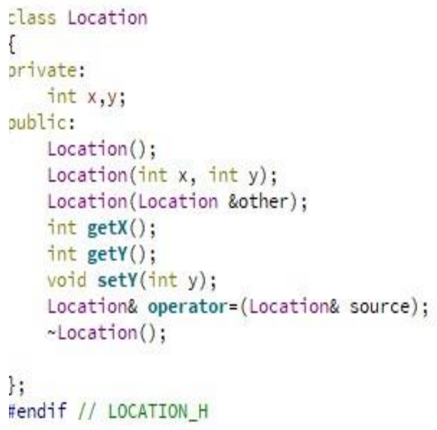
These screenshots are implementations of the getter, setter and function call operator overload of the member functions of the template class “MyCompare” and the functionality that these provide is that since the attributes of this class are encapsulated, these member functions allow a reference of the class to access and or modify them where necessary. Since the parameter variables (both implicit and explicit) are of template type it follows that these functions must be off template types as well and so this is why template functions were here.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely, provide a short explanation to support the claim
Not met		
Partially		

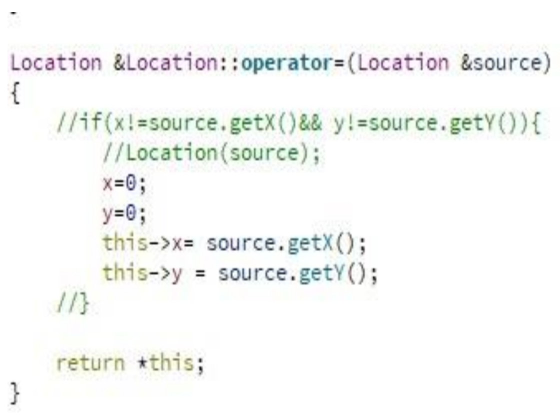
Completely	X	Objective of template functions is to provide the same functionality to variables of different types without the redundant naïve approach of a defining one for every specific variable or early binding for every specific case. And so by defining these functions as templates instead of any of the above mentioned options, we have satisfied the objective of template functions as we want the same operation to be executed regardless of variable type.

10. Operator Overloading

Screenshot: Location.h (left) and Assignment Operator Overloading Implementation (right)



```
class Location
{
private:
    int x,y;
public:
    Location();
    Location(int x, int y);
    Location(Location &other);
    int getX();
    int getY();
    void setY(int y);
    Location& operator=(Location& source);
    ~Location();

};
#endif // LOCATION_H
```



```
Location &Location::operator=(Location &source)
{
    //if(x!=source.getX()&& y!=source.getY()){
        //Location(source);
        x=0;
        y=0;
        this->x= source.getX();
        this->y = source.getY();
    //}

    return *this;
}
```

Motivation:
The class "Location" is another user-defined type that is used in the Bullet's move function as mentioned earlier. Seeing that it is a user defined type, to use it one had to overload the assignment operator for it as it is as seen in the screenshot above. The functionality and importance of this code is that it is passed as a reference parameter to the Bullet class (as mentioned earlier) that allows the bullet to compute the necessary trajectory.

How have you met the objectives?	Cross (X) the appropriate box	If you think that you have met the objective completely,
Not met		

Partially		**provide a short explanation to support the claim**
Completely	X	The objective of operator overloading is for the user to define his own desired implementation for that operator for that class, in this case the operator was assignment and so our own
		desired implementation was of the assignment operator was defined

Score Calculation

The game does score the user. This is done by incrementing a score number variable every time the user types in a letter corresponding to the one on the alien seen on screen. The variable is initialized to 0 at the start of a level and the more the user types in the letters correctly, the variable increments by one and is displayed to the user's screen. This variable is visible to the user throughout gameplay so he/she may keep track of their progress in that level.

There is also a number variable called "LIVES" visible to the user through-out gameplay. This is the number of lives or chances the user has before the user loses this level. Each Level has its initial number of lives according to the difficulty of the level, and is decremented by one each time an alien escapes the screen without the user successfully typing it in. Every-time it decrements, it is updated and displayed to the user accordingly. Once the value hits zero, the user loses this level and will have to replay it. If this variable is non-zero at the end of the time- limit, the user is promoted to the next level (unless this is the final level).

At the end of the level, a dialog user interface is instantiated with the user's final corrected score, accuracy and grade appears before the user to give them feedback on their skills. The grade is the overall assessor of the user's efficiency at this level and is computed by a formula that is directly proportional to the score obtained and inversely proportional to the number of lives lost by the user.

Multiplayer

This game does have a multiplayer option. In this mode, the game is open to two users. Since this is a typing game, the users cannot be typing at the same time so multiplayer feature allows users to alternate playing each level and then give them their statistics after both have played. The winner is decided by comparing their stats and determining whose are better. This comparison is done by the "MyCompare" template function object which is discussed above. There-after the users are given an option to either replay or proceed to the next level and test their skills against each other there.

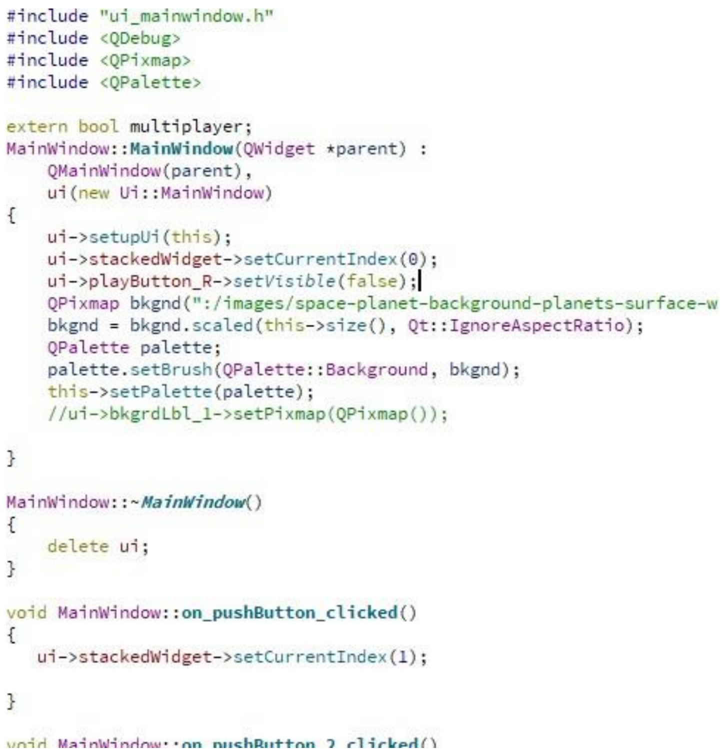
Additional Item

Does your game include:	Cross (X) the appropriate box
Graphical User Interface	**X**
Sound AND a help feature AND an about or credit section	
Other:	**X**

Option 1 –

Graphics

The game frame work used for this game was Qt Creator. Below is a snippet of the code that runs the main window graphics



```
#include "ui_mainwindow.h"
#include <QDebug>
#include <QPixmap>
#include <QPalette>

extern bool multiplayer;
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->stackedWidget->setCurrentIndex(0);
    ui->playButton_R->setVisible(false);
    QPixmap bkgnd(":/images/space-planet-background-planets-surface-w
    bkgnd = bkgnd.scaled(this->size(), Qt::IgnoreAspectRatio);
    QPalette palette;
    palette.setBrush(QPalette::Background, bkgnd);
    this->setPalette(palette);
    //ui->bkgrdLbl_1->setPixmap(QPixmap());

}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{
   ui->stackedWidget->setCurrentIndex(1);

}

void MainWindow::on_pushButton_2_clicked()
```

MainWindow.cpp Code Segment

[*What library or engine did you use? Provide screenshots of some graphics code. Just a segment of code is suffice.*]

Option 2 –

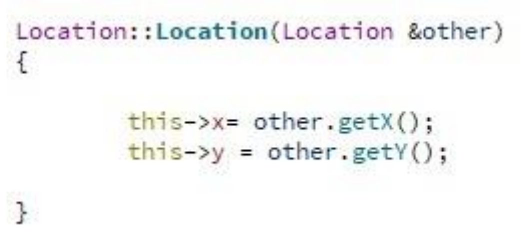
Sound, Support, and Credits

[*Provide some code of the different features? Just a segment of code is suffice.*]

Option 3 –
Other

Copy Constructor -

A copy constructor was defined for the class Location as Location variable needs to be passed into the Bullet Constructor as a parameter, so this automatically calls the copy constructor and thus hence a copy constructor was defined.

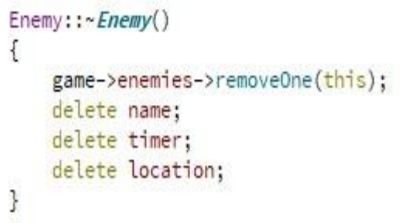


```
Location::Location(Location &other)
{

        this->x= other.getX();
        this->y = other.getY();

}
```

Location Class Copy Constructor

Destructors –

Destructors were defined for most of the user defined types but we will focus on the destructor for the class “Enemy”. This one is especially useful because every time an enemy collides with a bullet, it needs to be removed from the screen. This is easier said than done as the enemy needs to be removed from the Scene object in the user interface and also from the Vector containing all enemies on the screen and this would have been quite tedious if not for the destructor as all of this can be taken care of within the destructor which is called when as soon as it collides with a bullet.

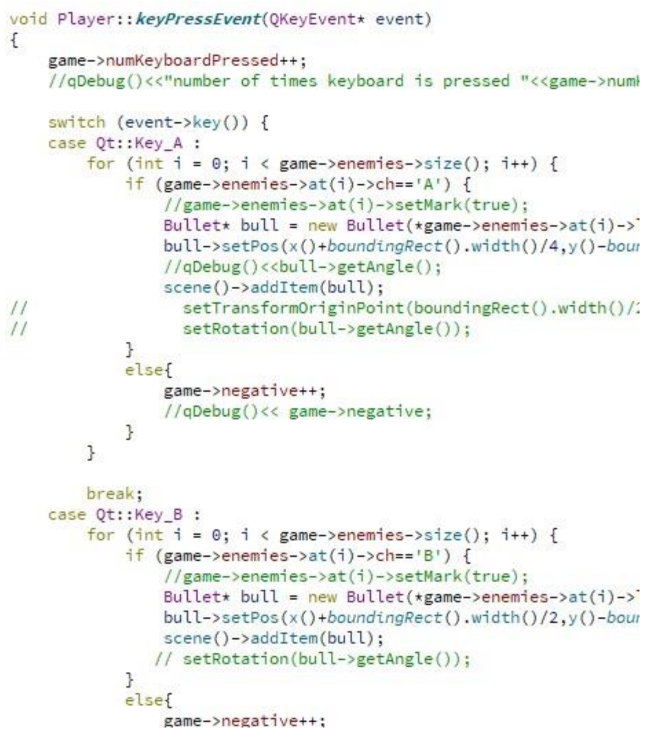


```
Enemy::~Enemy()
{
    game->enemies->removeOne(this);
    delete name;
    delete timer;
    delete location;
}
```

Enemy Class Destructor

Keyboard Events-

Keyboard event handlers were employed to keep track of which keys the user is typing in, are they the correct keys (depending on the characters on the screen) and then instantiate a bullet and score the user accordingly.

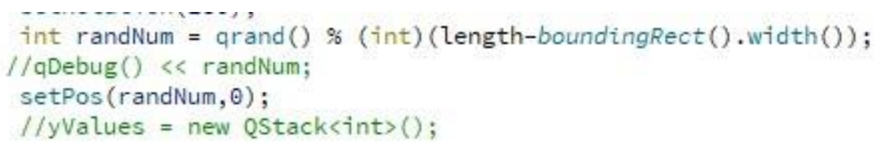


```
void Player::keyPressEvent(QKeyEvent* event)
{
    game->numKeyboardPressed++;
    //qDebug()<<"number of times keyboard is pressed "<<game->numk

    switch (event->key()) {
    case Qt::Key_A :
        for (int i = 0; i < game->enemies->size(); i++) {
            if (game->enemies->at(i)->ch=='A') {
                //game->enemies->at(i)->setMark(true);
                Bullet* bull = new Bullet(*game->enemies->at(i)->
                bull->setPos(x()+boundingRect().width()/4,y()-bour
                //qDebug()<<bull->getAngle();
                scene()->addItem(bull);
//                setTransformOriginPoint(boundingRect().width()/
//                setRotation(bull->getAngle());
            }
            else{
                game->negative++;
                //qDebug()<< game->negative;
            }
        }

        break;
    case Qt::Key_B :
        for (int i = 0; i < game->enemies->size(); i++) {
            if (game->enemies->at(i)->ch=='B') {
                //game->enemies->at(i)->setMark(true);
                Bullet* bull = new Bullet(*game->enemies->at(i)->
                bull->setPos(x()+boundingRect().width()/2,y()-bour
                scene()->addItem(bull);
               // setRotation(bull->getAngle());
            }
            else{
                game->negative++;
```

Event Handling

<u>Function Object-</u>

The function object “qrand()” was used to generate random numbers according to the desired range. They were used to generate random ascii values as discussed above and also used to generate random positions for the enemies(letters) to be created on the screen, increasing the user’s game-play experience as he/she will not know where to expect the next enemy to appear from. Not only that, this increase the user’s hand eye co-ordination as the user little time to spot the letter and type in a response, forcing him/her to increase his/her keyboard muscle memory



```
    int randNum = qrand() % (int)(length-boundingRect().width());
   //qDebug() << randNum;
    setPos(randNum,0);
    //yValues = new QStack<int>();
```

Random Number Generator Object

Appendix

[This is **OPTIONAL**: *Provide any additional information that you would like.*]