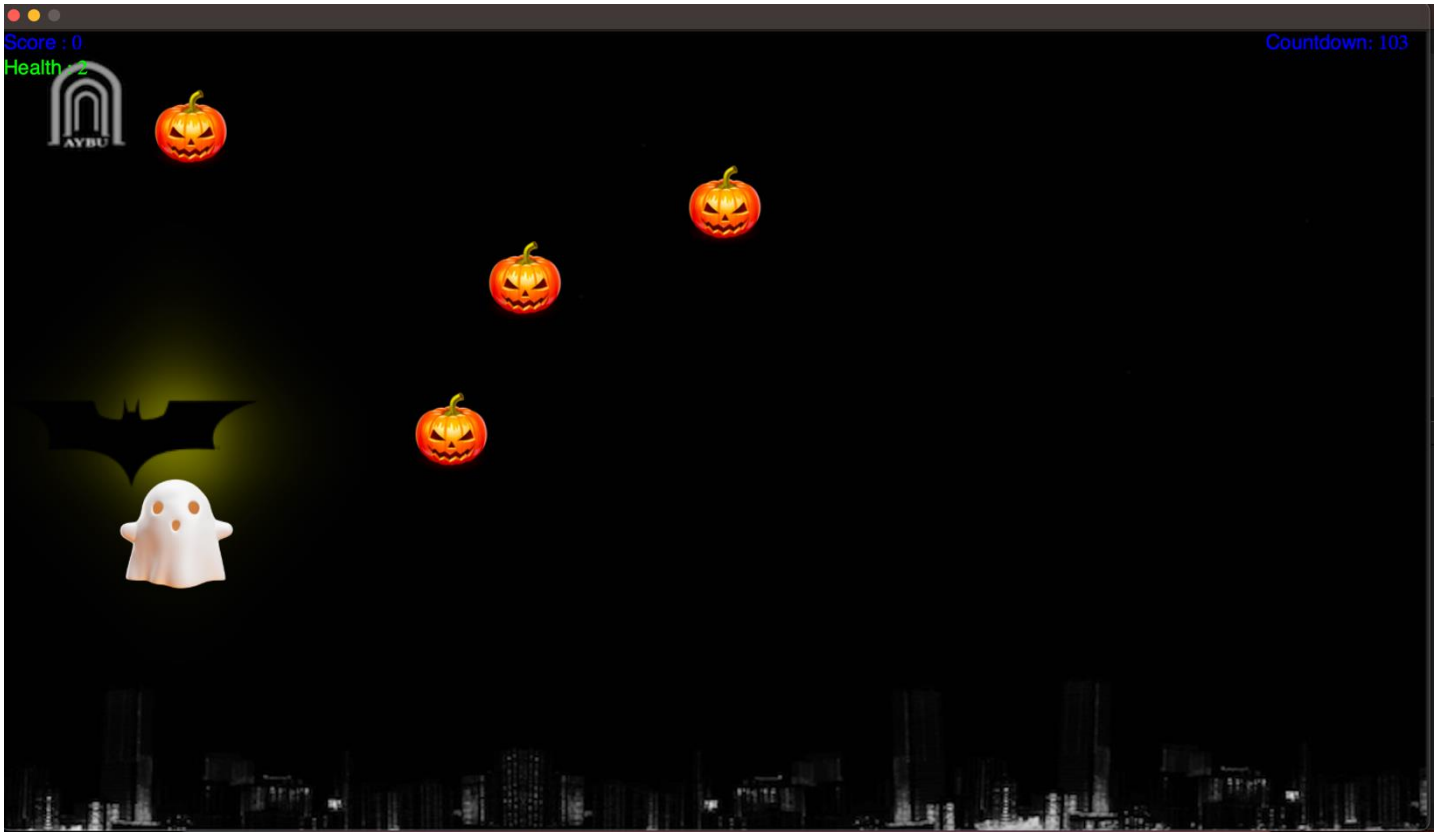
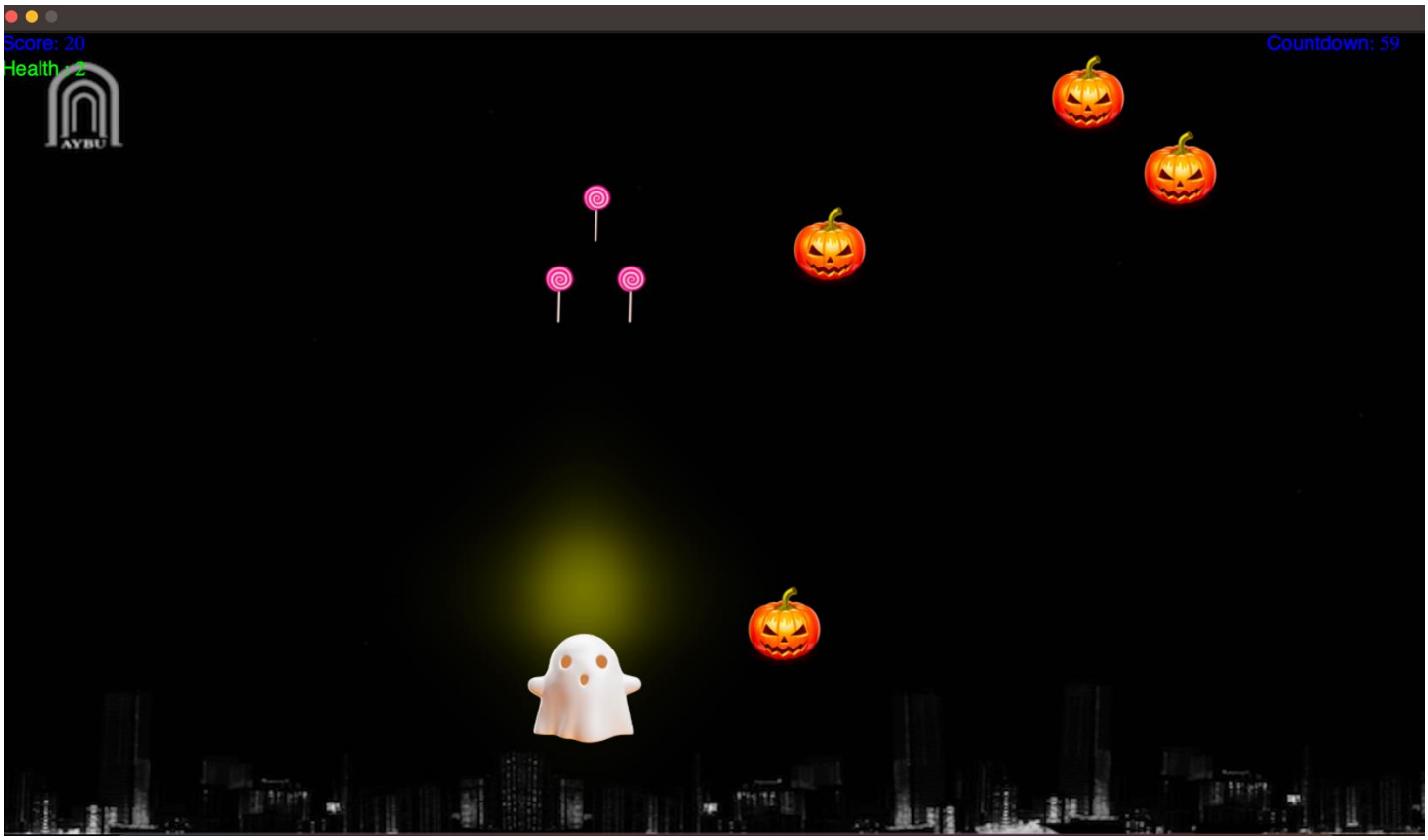
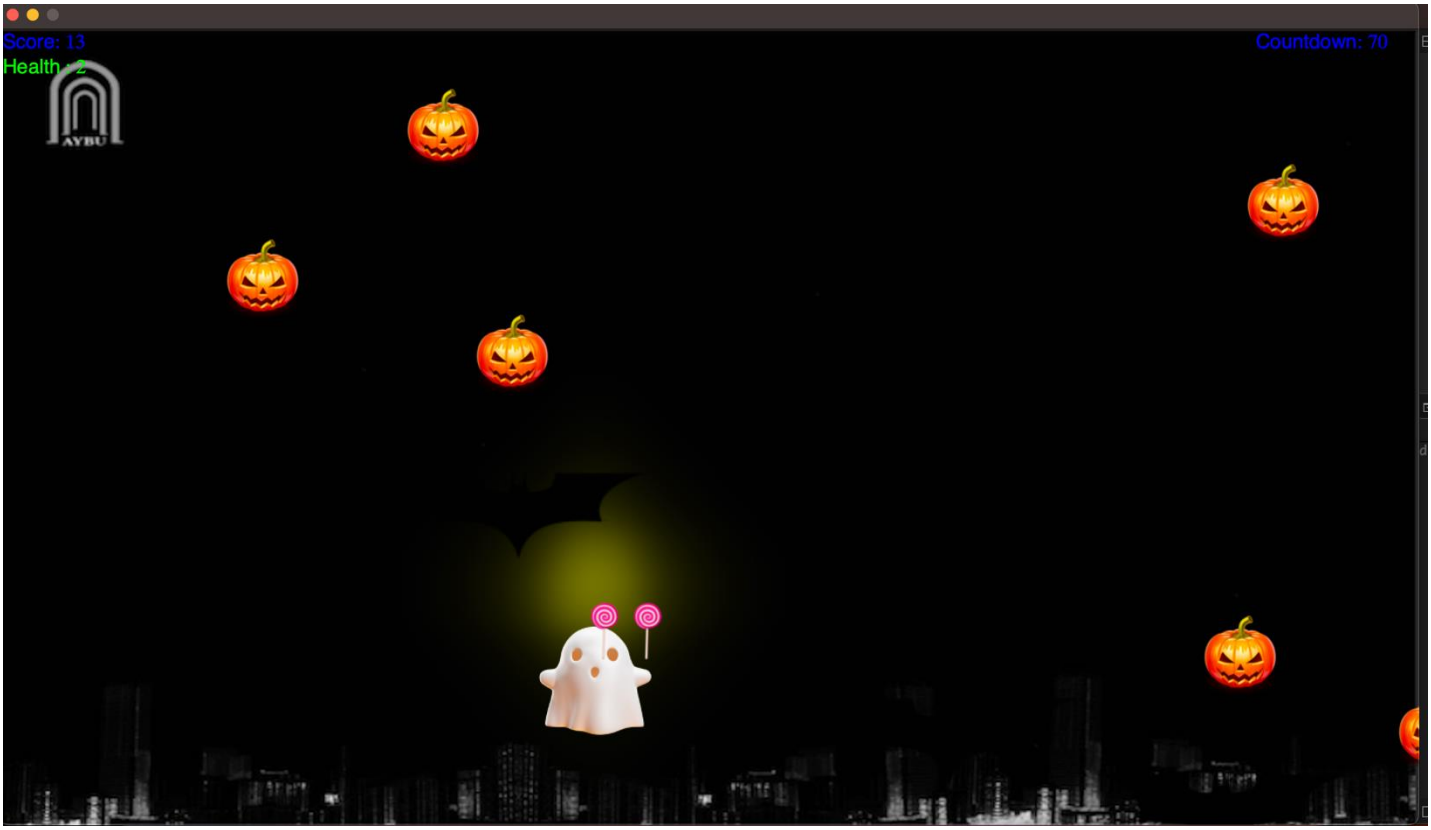
CENG206
PROGRAMMING LANGUAGES

GROUP 19
PROJECT-2 REPORT

20050111007 - UMUT ŞENOL
20050111082 - ŞEVVAL ÇOLAK
20050111024 - EDA AYAN
17050111042 - YUSUFHAN KELEŞ
21050111059 - SUDENUR KİBAROĞLU
20050111016 - EZGİ SUBAŞI


Score : 0
Countdown: 103


Score: 20
Countdown: 59


Score: 13
Health: 2
Countdown: 70
AYBU

Our C++ program must include the following programming features;

a. Inheritance
The EnemyChild class inherits Enemy and moves in a different plane by overriding the move function. It also differs in size.

EnemyChild:

```
#ifndef ENEMYCHILD_H                                    Not Committ
#define ENEMYCHILD_H

#include "Enemy.h"

class EnemyChild : public Enemy
{
    Q_OBJECT
public:
    explicit EnemyChild(QGraphicsItem *parent = nullptr);

    void move() override;

private:

};

#endif // ENEMYCHILD_H
```

```cpp
EnemyChild::EnemyChild(QGraphicsItem *parent) : Enemy(parent)
{
    // Farklı bir görsel atanabilir
    setPixmap(QPixmap(":/image/images/bat.png"));
}

void EnemyChild::move()
{
    // Çaprazdan hareket etmek için x ve y pozisyonlarını güncelle
    setPos(x() + 5, y() + 5);

    QList<QGraphicsItem *> colliding_items = collidingItems();
    for (QGraphicsItem *item : colliding_items) {
        if (typeid(*item) == typeid(Player)) {
            // Player ile çarpışma oldu
            // Sağlık azaltılabilir veya diğer işlemler yapılabilir
            game->health->decrease();
            // Enemy'yi sahneden kaldırma
            scene()->removeItem(this);
            delete this;
            return; // Çarpışma durumunda fonksiyondan çık
        }
    }

}
```

Enemy:

```cpp
#include <QDebug>
#include "Game.h"

extern Game *game;
Enemy::Enemy(QGraphicsItem *parent): QObject(), QGraphicsPixmapItem()
{   //set random position
    int random_number = rand() % 1200;
    setPos(random_number,0);

    //drew the rect
    setPixmap(QPixmap(":/image/images/pumpkin.png"));
    //Alaşağı ediyor fotoğrafı
    /*setTransformOriginPoint(50,50);
    setRotation(180);*/

    //connect
    QTimer * timer = new QTimer();
    connect(timer,SIGNAL(timeout() ) , this,SLOT(move()));

    timer -> start(200);
}
```

b. Exception handling;

```
void Enemy::move()

{
    try {
        QList<QGraphicsItem *> colliding_items = collidingItems();
        for (QGraphicsItem* item : colliding_items) {
            if (typeid(*item) == typeid(Player)) {
                // Player ile çarpışma oldu
                // Sağlık azaltılabilir veya diğer işlemler yapılabilir
                game->health->decrease();
                // Enemy'yi sahneden kaldırma
                scene()->removeItem(this);
                delete this;
                return; // Çarpışma durumunda fonksiyondan çık
            }
        }

        //move enemy down
        setPos(x(), y() + 5);

        if (pos().y() < 0) {
            //decrease the health
            game->health->decrease();
            scene()->removeItem(this);
            delete this;
        }
    } catch (std::exception& e) {
        // Hata durumunda yapılacak işlemler
        qDebug() << "Exception handled: " << e.what();
    } catch (...) {
        // Diğer tüm hata durumları için yapılacak işlemler
        qDebug() << "An unknown error has occurred";
    }
}
```

In the code example provided, the move() function is encapsulated within a try-catch block, which allows for the handling of exceptions or errors that might occur during the execution. This mechanism ensures that the program can gracefully handle unexpected scenarios and take appropriate actions.

When a collision occurs during the move operation, an exception of type CollisionException is thrown explicitly within the try block. This exception can be caught by the subsequent catch blocks, allowing for specialized handling of this specific type of error. For instance, if a CollisionException is caught, you can implement custom logic to handle the collision appropriately, such as adjusting the course of the object or notifying relevant parties.

On the other hand, if an exception of type std::exception& (a base class for most standard exceptions) is thrown within the try block, it can be caught by a dedicated catch block designed to handle std::exception& or its derived types. In such cases, you can utilize the e.what() function to obtain a description of the error provided by the exception object 'e'. This description can be useful for logging, displaying user-friendly error messages, or performing further error-specific actions.

To account for any other unforeseen errors or exceptions that are not explicitly caught by the previous catch blocks, a catch (...) block is included. The ellipsis (...) serves as a catch-all mechanism to handle any exception types that are not explicitly specified. While this catch-all block can capture unexpected errors, it's generally recommended to have specific catch blocks for known exceptions whenever possible, as it allows for more targeted error handling and better program control. By combining try-catch blocks with different exception types and specialized error handling, you can create a robust and resilient program that

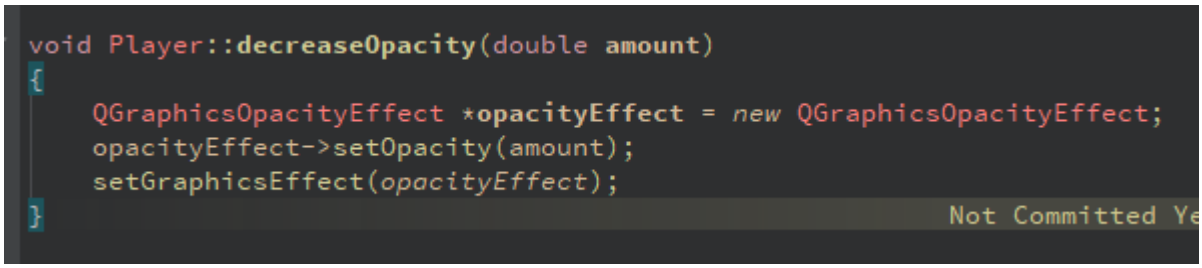
gracefully handles various scenarios, recovers from errors, and provides appropriate feedback to users or administrators.

c. Default parameter:

When our ghost is captured, there is a function that reduces its opacity. This function has a parameter called "amount" which determines the extent of the opacity reduction. If the "amount" parameter is not provided, the function uses a default value.

The default value for the "amount" parameter is chosen based on the desired behavior of the ghost when captured. It provides a convenient and customizable option for callers who don't need to specify a custom opacity reduction. By using a default parameter, the function becomes more flexible and user-friendly, reducing code repetition and simplifying function calls. It also promotes code organization and reusability, enabling consistent visual effects throughout the application.

In summary, the function to reduce ghost opacity when captured takes an optional "amount" parameter with a default value, allowing for customizable opacity reduction and easier function usage.



d. Operator overloading;

When the space key is pressed, two weapon objects are created and thrown randomly along the x-axis based on the player's position. We have overloaded the == operator for the weapon class to check if their positions coincide. If a coincidence is detected, both weapon objects are deleted from the scene. This allows for dynamic interaction between the player and the weapons, enhancing the gameplay experience. Overloading the == operator provides a concise and intuitive way to compare objects of the same class, improving code readability and maintainability.

```
} else if (event->key() == Qt::Key_Space) {
    //create a bulletwith a vector
    int randomX = rand() % 100; // 100, player genişliğine göre
    int randomX1 = rand() % 100 ;
    // Create the first bullet
    Bullet *bullet1 = new Bullet();
    bullet1->setPos(x()+ randomX1, y());
    scene()->addItem(bullet1);
    bullets.push_back(bullet1);  // Vector'e bullet1 ekleme

    // Create the second bullet with a slight offset to the right
    Bullet *bullet2 = new Bullet();
    bullet2->setPos(x() + randomX, y()); // Örnek olarak, 70 birim sağa kaydırıldı
    scene()->addItem(bullet2);
    bullets.push_back(bullet2);

    // Mermilerin çakışmasını kontrol etme
    if (*bullet1 == *bullet2) {
        // İki mermi aynı pozisyonda, her ikisini de listeden çıkart
        bullets.pop_back();
        scene()->removeItem(bullet1);
        delete bullet1;
        bullets.pop_back();
        scene()->removeItem(bullet2);
        delete bullet2;
}
}
```

```
bool Bullet::operator==(const Bullet& other) const
{
    return (pos() == other.pos());
}
```

e. Inline function;

In this code snippet, the `Game::gameOver()` function is declared as an inline function. Inline functions are functions that are expanded inline at the point of the function call, instead of being called as a separate function. By using the `inline` keyword, the compiler is instructed to insert the entire code of the `Game::gameOver()` function directly at the call site whenever the function is invoked. This can eliminate the overhead of a function call, resulting in potentially improved performance.

In this specific case, the `Game::gameOver()` function clears the current scene, creates a "Game Over" text item, sets its font and color, calculates its position, and adds it to the scene. The use of an inline function allows this logic to be efficiently expanded at the location where the function is called, eliminating the need for a separate function call overhead. Overall, inline functions can be beneficial when the function code is small and frequently called, as it can reduce the overhead of function calls and potentially improve performance.

```
inline void Game::gameOver()
{
    // Mevcut sahneyi temizleme
    scene->clear();

    // Game Over mesajını gösterme
    QGraphicsTextItem* gameOverText = new QGraphicsTextItem("Game Over");
    QFont gameOverFont("Arial", 50);
    gameOverText->setFont(gameOverFont);
    gameOverText->setDefaultTextColor(Qt::red);
    int xPos = width() / 2 - gameOverText->boundingRect().width() / 2;
    int yPos = height() / 2 - gameOverText->boundingRect().height() / 2;
    gameOverText->setPos(xPos, yPos);
    scene->addItem(gameOverText);
}
```

f. Constructor/destructor;

The constructor of the health class

```
Health :: Health (QGraphicsItem *parent) : QGraphicsTextItem(parent)
{
    //initialize the score to 0
    health = 2  ;

    //draw the text
    setPlainText(QString("Health : " + QString:: number(health) )); // Score : 0
    setDefaultTextColor(Qt::green);
    setFont(QFont("times", 20));
}                                                                    You 2023-05-
```

Deconstructor of the Player class

```
Player::~Player()
{
    // bullets vectorundeki tüm mermileri temizleme
    for (Bullet* bullet : bullets) {
        scene()->removeItem(bullet);
        delete bullet;
    }
    bullets.clear();                                          Not Com
}
```

g. Vectors :

```
#include <QGraphicsPixmapItem>
#include <QObject>
#include <QGraphicsItem>
#include <vector>  // vector için header dosyasını dahil edin

#include "Bullet.h"
class Player:public QObject, public QGraphicsPixmapItem{
    Q_OBJECT
public:
    Player(QGraphicsItem * parent=0);
    ~Player();

    void keyPressEvent(QKeyEvent * event);
public slots:
    void spawn();
    void spawn1();
    void decreaseOpacity(double amount);
private:
    std::vector<Bullet*> bullets;
};

#endif // PLAYER_H
```

```
    //create a bulletwith a vector
    int randomX = rand() % 100; // 100, player genişliğine göre
    int randomX1 = rand() % 100 ;
    // Create the first bullet
    Bullet *bullet1 = new Bullet();
    bullet1->setPos(x()+ randomX1, y());
    scene()->addItem(bullet1);
    bullets.push_back(bullet1);  // Vector'e bullet1 ekleme

    // Create the second bullet with a slight offset to the right
    Bullet *bullet2 = new Bullet();
    bullet2->setPos(x() + randomX, y()); // Örnek olarak, 70 birim sağa kaydır
    scene()->addItem(bullet2);
    bullets.push_back(bullet2);

    // Mermilerin çakışmasını kontrol etme
    if (*bullet1 == *bullet2) {
        // İki mermi aynı pozisyonda, her ikisini de listeden çıkart
        bullets.pop_back();
        scene()->removeItem(bullet1);
        delete bullet1;
        bullets.pop_back();
        scene()->removeItem(bullet2);
        delete bullet2;
```

h. Explicit heap dynamic variable;

```
Game::Game(QWidget* parent)
    : QGraphicsView(parent), timer0(nullptr){
    // we write code block  to create a scene
    QGraphicsScene * scene = new QGraphicsScene();      ⚠ Value stored to 'scene' dur


    // create the scene
    scene = new QGraphicsScene();                                          You 2023-05-1
    scene->setSceneRect(0,0,1200, 900); // make the scene 800x600 instead of infini

    // make the newly created scene the scene to visualize (since Game is a QGraphi
    // it can be used to visualize scenes)
    setScene(scene);
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setFixedSize(1200,900);

    //background.
    QPixmap backgroundImage(":image/images/background.jpg");
    QGraphicsPixmapItem* backgroundItem = new QGraphicsPixmapItem(backgroundImage);
    scene->addItem(backgroundItem);

    // create the player
    player = new Player();
    player-> setPixmap(QPixmap(":/image/images/ghost.png"));
```