CGL Mini Project
Space Invaders: Classic Atari Game

Our Team:

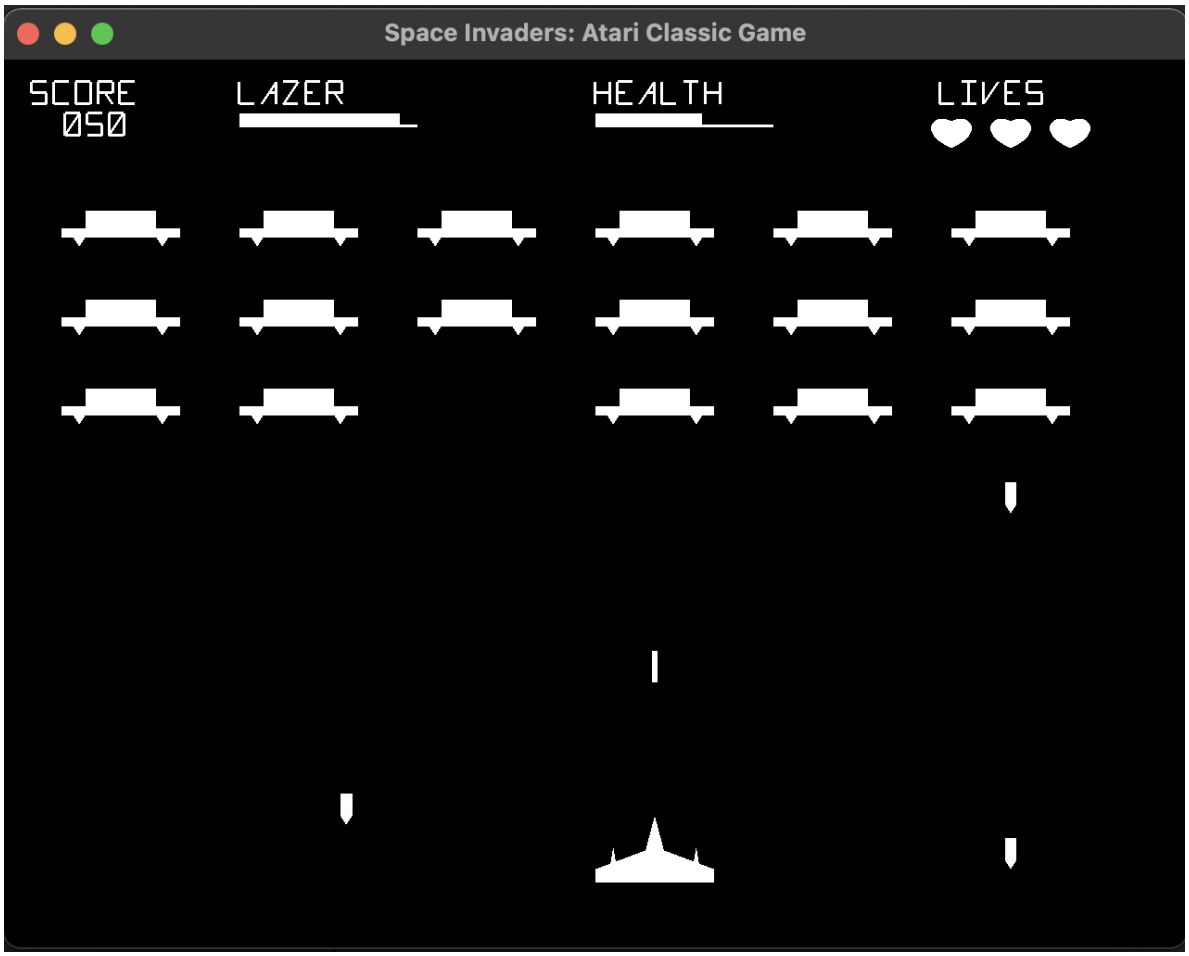
2110 - Sushant Behere
2113 - Atharva Bhosale
2121 - Nayan Choudhary
2129 - Mahesh Dudhe

Mentor:

Prof. Mrinali Bhajibhakre

Code below output images..

Output:




Space Invaders: Atari Classic Game
GAME
OVER
SCORE 150
PLAY AGAIN


Space Invaders: Atari Classic Game
YOU
WON
SCORE 900
PLAY AGAIN

Code:

Main.cpp

```
#include <GL/glew.h>
#define GL_SILENCE_DEPRECATION
#include <iostream>
#include <GLFW/glfw3.h>

#include "Player.hpp"
#include "Enemy.hpp"
#include "Bullet.hpp"
#include "GameOverScreen.hpp"
#include "Lazer.hpp"
#include "SegmentDiaplay.hpp"
#include <memory>
#include <vector>

//Make player obj
Player player = Player();

enum Game{ MENU_SCREEN = 0, GAME_PIAYING = 1, GAME_OVER = 2, GAME_INIT = 3};


std::vector<Enemy*> enemies = std::vector<Enemy*>();
std::vector<Bullet*> bullets = std::vector<Bullet*>();
std::vector<Lazer*> lazers = std::vector<Lazer*>();
std::vector<Heart*> hearts = std::vector<Heart*>();


HealthBar healthbar = HealthBar(0, 0.85);
int lazerManaCounter = 0;
int GameStatus = GAME_PIAYING;
int maxScore;
int currScore;
std::vector<int> score;
bool mouseOverReplayButton = false;
bool isMySonWinning = false;


std::vector<SegmentDisplay> LazerManaText;
std::vector<SegmentDisplay> Health;
std::vector<SegmentDisplay> LivesText;
std::vector<SegmentDisplay> ScoreText;
std::vector<SegmentDisplay> NumberText;
std::vector<SegmentDisplay> scoreText;
std::vector<SegmentDisplay> PlayAgainText;



void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    //std::cout<<key<<std::endl;

    if(key == GLFW_KEY_RIGHT){
        player.translateRight();
    }

    if(key == GLFW_KEY_LEFT){
        player.translateLeft();
    }

    if(key == GLFW_KEY_SPACE && action == GLFW_PRESS){
```

```
        if(lazerManaCounter ==100){
            lazers.push_back(new Lazer(player.posx, 0.85));
            lazerManaCounter = 0;
        }

    }
}

void initilizeGame(){

    GameStatus = GAME_PIAYING;

    enemies.clear();
    for(int i = 0; i < 6; i++){
        for(int j = 0; j < 3; j++){
            enemies.push_back(new Enemy(-0.8 + i*0.3, 0.6 - 0.2 * j));
        }
    }


    hearts.clear();
    player.setLives(3);
    player.isLiving = true;
    player.increaseHealth(100);
    std::cout<<player.getHealth()<<std::endl;
    healthbar.updateHealth(player);
    isMySonWinning = false;

    for(int i = 0; i < player.getLives(); i++){
        hearts.push_back(new Heart ( 0.6 + i * 0.1  , 0.83 ));
    }

    maxScore = enemies.size();
}


void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS){
        initilizeGame();
    }
}

static void cursor_position_callback(GLFWwindow* window, double xpos, double ypos)
{

    //std::cout<<"  x - "<<xpos<<", y - "<<ypos<<std::endl;

    if(xpos >= 169 && xpos <=523 && ypos >= 366 && ypos <= 424.5){
        mouseOverReplayButton = true;
    }else{
        mouseOverReplayButton = false;
    }


    if(mouseOverReplayButton){
        glfwSetMouseButtonCallback(window, mouse_button_callback);
    }

}



std::vector<int> numberToVec(int num){
    std::vector<int> ints;
```

```
    for(int i = 0; i < 3; i++ ){
        ints.push_back(num%10);
        num /= 10;
    }
    return ints;
}

void prepareTexts(){
    LazerManaText.push_back(SegmentDisplay("110000010000")); // L
    LazerManaText.push_back(SegmentDisplay("001100101100")); // A
    LazerManaText.push_back(SegmentDisplay("000010011100")); // Z
    LazerManaText.push_back(SegmentDisplay("110011010000")); // E
    LazerManaText.push_back(SegmentDisplay("110111100001")); // R

    Health.push_back(SegmentDisplay("111101100000")); // H
    Health.push_back(SegmentDisplay("110011010000")); // E
    Health.push_back(SegmentDisplay("001100101100"));  // A
    Health.push_back(SegmentDisplay("110000010000"));  // L
    Health.push_back(SegmentDisplay("0000100000001"));  //T
    Health.push_back(SegmentDisplay("111101100000")); // H

    LivesText.push_back(SegmentDisplay("110000010000")); // L
    LivesText.push_back(SegmentDisplay("0000100100001")); // I
    LivesText.push_back(SegmentDisplay("110000001100")); // V
    LivesText.push_back(SegmentDisplay("110011010000")); // E
    LivesText.push_back(SegmentDisplay("011011110000")); // S

    ScoreText.push_back(SegmentDisplay("011011110000")); // S
    ScoreText.push_back(SegmentDisplay("110010010000")); // C
    ScoreText.push_back(SegmentDisplay("111110010000")); // O
    ScoreText.push_back(SegmentDisplay("110111100001")); // R
    ScoreText.push_back(SegmentDisplay("110011010000")); // E

    PlayAgainText.push_back(SegmentDisplay("110111100000")); // P
    PlayAgainText.push_back(SegmentDisplay("110000010000")); //L
    PlayAgainText.push_back(SegmentDisplay("001100101100"));  // A
    PlayAgainText.push_back(SegmentDisplay("000000001110")); // Y
    PlayAgainText.push_back(SegmentDisplay("000000000000")); // _
    PlayAgainText.push_back(SegmentDisplay("001100101100"));  // A
    PlayAgainText.push_back(SegmentDisplay("111010110000")); // G
    PlayAgainText.push_back(SegmentDisplay("001100101100"));  // A
    PlayAgainText.push_back(SegmentDisplay("0000100100001")); // I
    PlayAgainText.push_back(SegmentDisplay("111100000011"));  // N


    NumberText.push_back(SegmentDisplay("111110011100")); // 0
    NumberText.push_back(SegmentDisplay("001100000000")); // 1
    NumberText.push_back(SegmentDisplay("100111110000")); // 2
    NumberText.push_back(SegmentDisplay("001111110000")); // 3
    NumberText.push_back(SegmentDisplay("011101100000")); // 4
    NumberText.push_back(SegmentDisplay("011011110000")); // 5
    NumberText.push_back(SegmentDisplay("111011110000")); // 6
    NumberText.push_back(SegmentDisplay("001110000000")); // 7
    NumberText.push_back(SegmentDisplay("111111110000")); // 8
    NumberText.push_back(SegmentDisplay("011111110000")); // 9

}




int main(int argc, const char * argv[]) {


    prepareTexts();
```

```
//init glfw and window
if (!glfwInit())
{
    std::cout << "glfw init failed";
}

GLFWwindow* window = glfwCreateWindow(640, 480, "Space Invaders: Atari Classic Game", NULL, NULL);
if (!window)
{
    std::cout << "window init failed";
}

glfwMakeContextCurrent(window);
glfwSetKeyCallback(window, key_callback);
glfwSetTime(0);


//enemies.push_back(new Enemy(0,0.8));

initilizeGame();


int frameCounter = 0;
int randTime = 3;


GameOverScreen gameOverScreen = GameOverScreen();


bool gameInit = false;
scoreText.push_back(SegmentDisplay(NumberText[0]));
scoreText.push_back(SegmentDisplay(NumberText[0]));
scoreText.push_back(SegmentDisplay(NumberText[0]));

//SegmentDisplay letter = SegmentDisplay("011011110000"); //L



while (!glfwWindowShouldClose(window))
{

    glClear(GL_COLOR_BUFFER_BIT);




    //get mouse and keyboard events
    glfwPollEvents();

    //mouse events callback
    //glfwSetMouseButtonCallback(window, mouse_button_callback);
    glfwSetCursorPosCallback(window, cursor_position_callback);


    switch(GameStatus){

        case GAME_PIAYING:
        {
            // Keep running
            frameCounter++;

            gameInit = false;
```

```
healthbar.draw();


for(int i = 0; i < Health.size(); i++){
    Health[i].draw(3, 0 + i*0.037, 0.9, 0.5);
}
for(int i = 0; i < LivesText.size(); i++){
    LivesText[i].draw(3, 0 + i*0.037 + 0.58 , 0.9, 0.5);
}
for(int i = 0; i < LazerManaText.size(); i++){
    LazerManaText[i].draw(3, 0 + i*0.037 + -0.6 , 0.9, 0.5);
}

for(int i = 0; i < ScoreText.size(); i++){
    ScoreText[i].draw(3, 0 + i*0.037 + -0.95 , 0.90, 0.5);
}

for(int i = 0; i < scoreText.size(); i++){
    scoreText[i].draw(3, 0 + i*(-0.037) + -0.85 , 0.83, 0.5);
}



//draw enemies
for(Enemy *enemy: enemies){
    enemy->draw();
}

for(Lazer *lazer: lazers){
    if(lazer->active){
        lazer->draw();
    }

}

if(lazerManaCounter != 100){
    if(frameCounter%20 == 19){
        if(lazerManaCounter + 10 > 100){
            lazerManaCounter = 100;
        }else{
            lazerManaCounter += 10;
        }
    }
}

//drawLazerMana
glBegin(GL_POLYGON);
glVertex2f(-0.6 , 0.85);
glVertex2f(((float)lazerManaCounter/100)*(0.3) - 0.6 , 0.85 );
glVertex2f(((float)lazerManaCounter/100)*(0.3) - 0.6 , 0.88 );
glVertex2f(-0.6 , 0.88 );
glEnd();
glBegin(GL_LINES);
glVertex2f(-0.6 , 0.85);
glVertex2f((0.3) - 0.6 , 0.85 );
glEnd();




int bulletIndex = 0;
for(Bullet *bullet: bullets){
    bullet->draw();

    if(bullet->destroyPlayerIfNear(player, hearts, healthbar) == -1){
```

```
            bullets.erase(bullets.begin()+bulletIndex);
        }else if (bullet->destroyPlayerIfNear(player, hearts, healthbar) == -2){
            player.isLiving = false;
            player.~Player();
            GameStatus = GAME_OVER;
        };
        bulletIndex++;

    }

    //draw player
    if(player.isLiving){
        player.draw();
    }

    for(Heart* heart: hearts){
        heart-> draw();
    }

    //std::cout<<hearts.size()<<std::endl;

    //bullet frequency
    int random = rand()%2;


    //for moving bullets (bullet speed)
    if(frameCounter%20 == 19){
        for(Bullet *bullet: bullets){
            bullet->moveDown();
        }

    }

    if(frameCounter%20 == 19){

        for(int k = 0; k < lazers.size(); k++){
            if(lazers[k]->active){
                lazers[k]->destroyEnemyIfNear(enemies);
            }
            if(lazers[k]->posy > 1){  //memory management
                lazers.erase(lazers.begin()+k);
            }else{
                lazers[k]->moveUp();
            }
        }
        //std::cout<<lazers.size()<<std::endl;

        currScore = (maxScore - enemies.size())*50;
        score = numberToVec(currScore);
        scoreText.clear();
        for(int i = 0; i < score.size(); i++){
            //std::cout<<NumberText[score[i]].letter<<std::endl;
            scoreText.push_back(SegmentDisplay(NumberText[score[i]]));
        }
        //std::cout<<std::endl;

    }


    //for moving enemies and for firing bullets
    if(frameCounter == 200){

        for(Enemy *enemy: enemies){
            enemy->move();
        }
```

```
            frameCounter = 0;
            randTime--;
            if(randTime <= 0){
                randTime = random;
                //std::cout<<random<<std::endl;
                int rand1 = (rand())%enemies.size();
                if(enemies.size()!=0){
                    bullets.push_back(new Bullet(enemies[rand1]->posx,enemies[rand1]->posy));
                }

            }
        }




        //delete fargone bullets as memory management
        for(int y  = 0; y < bullets.size(); y++){
            if(bullets[y]->posy < -1.5){
                bullets[y]->~Bullet();
                bullets.erase(bullets.begin() + y);
            }
        }

        if(enemies.size()==0){
            isMySonWinning = true;
            GameStatus = GAME_OVER;
        }

        break;

    }

    case GAME_OVER:{

        bullets.clear();
        enemies.clear();




        if(mouseOverReplayButton){
            float x = 0.03;
            float y = 0.04;

            glLineWidth(5);
            glBegin(GL_LINES);
            glVertex2f(-0.5 + x  , -0.57 + y);
            glVertex2f(0.6 + x,-0.57 + y);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(0.6 + x,-0.57 + y);
            glVertex2f(0.6 + x,-0.8 + y);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(0.6 + x,-0.8 + y);
            glVertex2f(-0.5 + x,-0.8 + y);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(-0.5 + x, -0.57 + y);
            glVertex2f(-0.5 + x,-0.8 + y);
```

```
            glEnd();

        }else{

            float x = 0.03;
            float y = -0.03;
            float boxSize = 0.9;

            glLineWidth(5);
            glBegin(GL_LINES);
            glVertex2f((-0.5)*boxSize + x  , (-0.57)*boxSize + y);
            glVertex2f(0.6*boxSize + x,-0.57*boxSize + y);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(0.6*boxSize + x,-0.57*boxSize + y);
            glVertex2f(0.6*boxSize + x,-0.8*boxSize + y);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(0.6*boxSize + x,-0.8*boxSize + y);
            glVertex2f(-0.5*boxSize + x,-0.8*boxSize + y);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(-0.5*boxSize + x, -0.57*boxSize + y);
            glVertex2f(-0.5*boxSize + x,-0.8*boxSize + y);
            glEnd();

        }


        gameOverScreen.draw(ScoreText, scoreText, PlayAgainText, mouseOverReplayButton, isMySonWinning);

        break;
      }


      case GAME_INIT:{

        //if(!gameInit){
            //gameInit = true;
        //}
      }
    }
    glfwSwapBuffers(window);
  }

  glfwTerminate();

  return 0;
}
```

Player.hpp

```cpp
#ifndef Player_hpp
#define Player_hpp

#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>

class Player{

private:
    int lives = 3;
    int health = 100;

public:
    float posx;
    bool isLiving = true;

    void draw();
    void translateLeft();
    void translateRight();
    void decreaseLives();
    void decreaseHealth();
    void increaseHealth(int powerUp);
    int getLives();
    void setLives(int);
    int getHealth();
    ~Player();

};

#endif /* Player_hpp */
```

Player.cpp

```cpp
#define GL_SILENCE_DEPRECATION
#include "Player.hpp"

void Player::draw(){

    //draw triangle
    glBegin(GL_POLYGON);
    glVertex2f(0.1 + posx, -0.82);
    glVertex2f(0.1 + posx, -0.85);
    glVertex2f(-0.1 + posx, -0.85);
    glVertex2f(-0.1 + posx, -0.82);
    glVertex2f(0 + posx, -0.77);
    glEnd();
```

```
    glBegin(GL_TRIANGLES);
    glVertex2f(-0.080 + posx, -0.85);
    glVertex2f(-0.060 + posx, -0.85);
    glVertex2f(-0.070 + posx, -0.77);
    glEnd();

    glBegin(GL_TRIANGLES);
    glVertex2f(0.080 + posx, -0.85);
    glVertex2f(0.060 + posx, -0.85);
    glVertex2f(0.070 + posx, -0.77);
    glEnd();

    glBegin(GL_TRIANGLES);
    glVertex2f(0.03 + posx, -0.85);
    glVertex2f(-0.03 + posx, -0.85);
    glVertex2f(0 + posx, -0.70);
    glEnd();
}

void Player::translateLeft(){

    if(!(posx < (-0.8))){
        posx -= 0.05;
        return;
    }
    posx = -0.8;
    //std::cout<<posx<<std::endl;
}

void Player::translateRight(){

    if(!(posx > 0.8)){
        //std::cout<<"---"<<posx<<std::endl;
        posx += 0.05;
        return;
    }
    posx = 0.8;
    //std::cout<<posx<<std::endl;
}

void Player::decreaseLives(){
    lives--;
}

void Player::decreaseHealth(){
    health -= 40;
}

int Player::getLives(){
    return lives;
}

void Player::setLives(int newLife){
    lives = newLife;
}

int Player::getHealth(){
    return health;
}

void Player::increaseHealth(int powerUp){
    if(health <= 0 ){
        health = powerUp;
        return;
```

```
    }
    if(health + powerUp >= 100){
        health = 100;
        return;
    }
    health+=powerUp;
}

Player::~Player(){

}
```

Enemy.hpp

```
#ifndef Enemy_hpp
#define Enemy_hpp

#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>



class Enemy{

private:
    int health = 100;

public:
    float posx;
    float posy;
    int dir = 1;

    void draw();
    void translateLeft();
    void translateRight();
    void decreaseHealth();
    int getHealth();
    void move();
    Enemy(float, float);

};
#endif /* Enemy_hpp */
```

Enemy.cpp

```
//
//  Enemy.cpp
//  AtariSpaceInvaders
//
//  Created by Atharva Bhosale on 20/11/21.
//

#define GL_SILENCE_DEPRECATION
#include "Enemy.hpp"


void Enemy::draw(){

    glBegin(GL_POLYGON);
    glVertex2f(-0.06 + posx, 0.02 + posy);
    glVertex2f(-0.1 + posx, 0.02 + posy);
```

```
    glVertex2f(-0.1 + posx, 0 + posy);
    glVertex2f(0.1 + posx, 0 + posy);
    glVertex2f(0.1 + posx, 0.02 + posy);
    glVertex2f(0.06 + posx, 0.02 + posy);
    glVertex2f(0.06 + posx, 0.06 + posy);
    glVertex2f(-0.06 + posx, 0.06 + posy);
    glVertex2f(-0.06 + posx, 0.02 + posy);
    glEnd();


    glBegin(GL_TRIANGLES);
    glVertex2f(-0.08 + posx, 0 + posy);
    glVertex2f(-0.06 + posx, 0 + posy);
    glVertex2f(-0.07 + posx, -0.02 + posy);
    glEnd();

    glBegin(GL_TRIANGLES);
    glVertex2f(0.08 + posx, 0 + posy );
    glVertex2f(0.06 + posx, 0 + posy);
    glVertex2f(0.07 + posx, -0.02 + posy);
    glEnd();
}

void Enemy::move(){

    dir = -1*dir;

    if(dir == -1){
        posx += 0.08;
    }else{
        posx -= 0.08;
    }
}



Enemy::Enemy(float posx, float posy){
    this->posx = posx;
    this->posy = posy;
}
```

Bullet.hpp

```
#ifndef Bullet_hpp
#define Bullet_hpp


#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>
#include "Player.hpp"
#include "Heart.hpp"
#include "HealthBar.hpp"
#include <vector>
#include <cmath>

class Bullet{
public:
    float posx;
    float posy;

    Bullet(float, float);
```

```
    ~Bullet();
    void draw();
    void moveDown();
    int destroyPlayerIfNear(Player&, std::vector<Heart*>&, HealthBar&);

};

#endif /* Bullet_hpp */
```

Bullet.cpp

```


#define GL_SILENCE_DEPRECATION
#include "Bullet.hpp"

void Bullet::draw(){

    //draw triangle
    glBegin(GL_POLYGON);
    glVertex2f(0.01 + posx, 0.05 + posy);
    glVertex2f(0.01 + posx, 0 + posy);
    glVertex2f(0 + posx, -0.02 + posy);
    glVertex2f(-0.01 + posx, 0 + posy);
    glVertex2f(-0.01 + posx, 0.05 + posy);
    glEnd();
}

Bullet::Bullet(float posx, float posy){
    this->posx = posx;
    this->posy = posy;
}

void Bullet::moveDown(){
    posy -= 0.05;
}

Bullet::~Bullet(){
    //
    //std::cout<<"bullet distruct - "<<std::endl;;
}

int Bullet::destroyPlayerIfNear(Player& player, std::vector<Heart*>& hearts, HealthBar& healthBar){

    float distanceBtw = sqrt(pow((posx - player.posx),2) + pow((posy + 0.85), 2));
    //std::cout<<"dist btw - "<<distanceBtw<<std::endl;

    if(distanceBtw < 0.15){
        if(player.getHealth() < 0){
            if(player.getLives() == 0){
                return -2;
                }else{
                    player.decreaseLives();
                    hearts.pop_back();
                    //std::cout<<"heart array size "<<std::endl;;
                }
            player.increaseHealth(100);
            return -1;
            }else{
                if(player.getHealth() - 20 <= 0){
                    if(player.getLives() - 1 == 0){
                        return -2;
                    }
```

```
                player.decreaseLives();
                hearts.pop_back();
                player.increaseHealth(100);
                healthBar.updateHealth(player);

            }else{
                player.decreaseHealth();
                healthBar.updateHealth(player);
            }

            //std::cout<<"player lif - "<<player.getLives()<<"-- health "<<player.getHealth()<<std::endl;;
            return -1;
        }
    }
    return 0;
}
```

GameOverScreen.hpp

```
#ifndef GameOverScreen_hpp
#define GameOverScreen_hpp

#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>
#include "SegmentDiaplay.hpp"
#include <vector>

class GameOverScreen{
public:
    void draw(std::vector<SegmentDisplay>&,std::vector<SegmentDisplay>&, std::vector<SegmentDisplay>&, bool, bool);
};

#endif /* GameOverScreen_hpp */
```

GameOverScreen.cpp

```
#define GL_SILENCE_DEPRECATION
#include "GameOverScreen.hpp"

void GameOverScreen::draw(std::vector<SegmentDisplay>& ScoreText, std::vector<SegmentDisplay>& scoreText, std::vector<SegmentDisplay>&
playAgainText, bool onPlayAgainHover, bool isMySonWinning){

    std::vector<SegmentDisplay> GameOverText;
    std::vector<SegmentDisplay> GameOverText2;
    std::vector<SegmentDisplay> YouText;
    std::vector<SegmentDisplay> WonText;



    GameOverText.push_back(SegmentDisplay("111010110000")); // G
    GameOverText.push_back(SegmentDisplay("001100101100")); // A
    GameOverText.push_back(SegmentDisplay("111100001010")); // M
    GameOverText.push_back(SegmentDisplay("1100011010000")); // E
    //GameOverText.push_back(SegmentDisplay("000000000000")); // _
    GameOverText2.push_back(SegmentDisplay("111110010000")); // O
    GameOverText2.push_back(SegmentDisplay("110000001100")); // V
    GameOverText2.push_back(SegmentDisplay("1100011010000")); // E
    GameOverText2.push_back(SegmentDisplay("110111100001")); // R

    YouText.push_back(SegmentDisplay("000000001110")); // Y
    YouText.push_back(SegmentDisplay("111110010000")); // O
```

```
    YouText.push_back(SegmentDisplay("111100010000")); // U
    //
    WonText.push_back(SegmentDisplay("111100000101")); // W
    WonText.push_back(SegmentDisplay("111110010000")); // O
    WonText.push_back(SegmentDisplay("111100000011"));  // N




    //std::cout<<" game won "<<isMySonWinning<<std::endl;


    if(!isMySonWinning){
        for(int i = 0; i < GameOverText.size(); i++){
                GameOverText[i].draw(15, -0.4 + i*0.25, 0.5, 3);
            }
            for(int i = 0; i < GameOverText2.size(); i++){
                GameOverText2[i].draw(15, -0.4 + i*0.25, 0.1, 3);
            }
    }else{
        for(int i = 0; i < YouText.size(); i++){
                YouText[i].draw(15, -0.3 + i*0.25, 0.5, 3);
            }
            for(int i = 0; i < WonText.size(); i++){
                WonText[i].draw(15, -0.3 + i*0.25, 0.1, 3);
            }
    }

    for(int i = 0; i < ScoreText.size(); i++){
        ScoreText[i].draw(10, -0.6 + i*0.15 , -0.3, 1.5);
    }

    for(int i = 0; i < scoreText.size(); i++){
        scoreText[i].draw(10, i*(-0.15) +0.7 , -0.3, 1.5);
    }

    for(int i = 0; i < playAgainText.size(); i++){
        if(onPlayAgainHover){
            playAgainText[i].draw(7, i*(0.1) + -0.4 , -0.7, 1);
        }else{
            playAgainText[i].draw(5, i*(0.09) + -0.35 , -0.675, 0.7);
        }
    }




}
```

Lazer.hpp

```
#ifndef Lazer_hpp
#define Lazer_hpp

#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>
#include "Enemy.hpp"
#include <cmath>
#include <vector>

class Lazer{
public:
```

```
    float posx;
    float posy;
    bool active = true;

    Lazer(float, float);
    ~Lazer();
    void draw();
    void moveUp();
    int destroyEnemyIfNear(std::vector<Enemy*>&);

};
#endif /* Lazer_hpp */
```

Lazer.cpp

```
#define GL_SILENCE_DEPRECATION
#include "Lazer.hpp"

void Lazer::draw(){

    //draw triangle
    glBegin(GL_POLYGON);
    glVertex2f(0.005 + posx, 0.07 + posy);
    glVertex2f(0.005 + posx, 0 + posy);
    glVertex2f(-0.005 + posx, 0 + posy);
    glVertex2f(-0.005+ posx, 0.07 + posy);
    glEnd();

}

Lazer::Lazer(float posx, float posy){
    this->posx = posx;
    this->posy = -0.80;
}

void Lazer::moveUp(){
    posy += 0.05;
}

Lazer::~Lazer(){
    //
    std::cout<<"bullet distruct - "<<std::endl;;
}

int Lazer::destroyEnemyIfNear(std::vector<Enemy*>& enemies){

    for(int i = 0; i< enemies.size(); i++){

        float distanceBtw = sqrt(pow((posx - enemies[i]->posx),2) + pow((posy - enemies[i]->posy), 2));



            if(distanceBtw < 0.12){
                enemies.erase(enemies.begin() + i);
                this->active = false;
                return -1;
            }
    }
    //std::cout<<std::endl<<std::endl;

    return 0;
```

```
}
```

Heart.hpp

```
#define GL_SILENCE_DEPRECATION
#ifndef Heart_hpp
#define Heart_hpp

#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>
#include <cmath>

class Heart {
public:
    float posx;
    float posy;

    Heart (float, float);
    void draw();
};



#endif /* Heart_hpp */
```

Heart.cpp

```
#include "Heart.hpp"


void Heart::draw(){

    glPointSize(1);
    //glColor3ub(255, 0, 0);  // Color Red
    glBegin(GL_POLYGON);
        for (float x = -1.139; x <= 1.139; x += 0.001)
        {
            float delta = cbrt(x*x) * cbrt(x*x) - 4*x*x + 4;
            float y1 = (cbrt(x*x) + sqrt(delta)) / 2;
            float y2 = (cbrt(x*x) - sqrt(delta)) / 2;
            glVertex2f(0.03*x + posx, 0.03*y1 + posy);
            glVertex2f(0.03*x + posx, 0.03*y2 + posy);
        }
    glEnd();

}

Heart::Heart(float posx, float posy){
    this->posx = posx;
    this->posy = posy;
}
```

HealthBar.hpp

```
#ifndef HealthBar_hpp
#define HealthBar_hpp

#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>
#include "Player.hpp"
```

```
class HealthBar{
public:
    float posx;
    float posy;
    int health = 100;

    void updateHealth(Player&);
    void draw();
    HealthBar(float, float);
};


#endif /* HealthBar_hpp */
```

Healthbar.cpp

```
#define GL_SILENCE_DEPRECATION
#include "HealthBar.hpp"

void HealthBar::draw(){
    glBegin(GL_POLYGON);
    glVertex2f(0 + posx, 0 + posy);
    glVertex2f(((float)health/100)*(0.3) + posx, 0 + posy);
    glVertex2f(((float)health/100)*(0.3) + posx, 0.03 + posy);
    glVertex2f(0 + posx, 0.03 + posy);
    glEnd();

    glLineWidth(3);
    glBegin(GL_LINES);
    glVertex2f(0 + posx, 0 + posy);
    glVertex2f((0.3) + posx, 0 + posy);    glEnd();
    glEnd();

    //std::cout<<(((float)health/100))*(0.3)<<"- "<<this->health<<std::endl;
}


void HealthBar::updateHealth(Player& player){

    this->health = player.getHealth();

}

HealthBar::HealthBar(float posx, float posy){
    this->posx = posx;
    this->posy = posy;
}
```

SegmentDiaplay.hpp

```
#define GL_SILENCE_DEPRECATION
#ifndef SegmentDiaplay_hpp
#define SegmentDiaplay_hpp

#include <stdio.h>
#include <iostream>
#include <GLFW/glfw3.h>
//#include <String>

class SegmentDisplay{

public:
    float posx;
    float posy;
```

```
    std::string letter;

    void draw(int,float , float, float);
    SegmentDisplay(std::string);

};

#endif /* SegmentDiaplay_hpp */
```

SegmentDiaplay.cpp

```


#include "SegmentDiaplay.hpp"

void SegmentDisplay::draw(int width, float posx, float posy, float size){

//-------2vertical
    glLineWidth(width);

    if(letter[0] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0 + posx, 0 + posy);
            glVertex2f(0 + posx, 0.05*size + posy);
            glEnd();
    }

    if(letter[1] == '1'){
        glBegin(GL_LINES);
        glVertex2f(0 + posx, 0.05*size + posy);
        glVertex2f(0 + posx, 0.1*size + posy);
        glEnd();
    }

//---------2nd 2 vertical
    if(letter[2] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0.05*size + posx, 0 + posy);
            glVertex2f(0.05*size + posx, 0.05*size + posy);
            glEnd();
    }

    if(letter[3] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0.05*size + posx, 0.05*size + posy);
            glVertex2f(0.05*size + posx, 0.1*size + posy);
            glEnd();
    }


//----------middle 4 horizontal

    if(letter[4] == '1'){
        glBegin(GL_LINES);
        glVertex2f(0 + posx, 0.1*size + posy);
        glVertex2f(0.05*size + posx, 0.1*size + posy);
        glEnd();
    }

    if(letter[5] == '1'){
        glBegin(GL_LINES);
        glVertex2f(0 + posx, 0.05*size + posy);
        glVertex2f(0.025*size + posx, 0.05*size + posy);
```

```
        glEnd();
    }

    if(letter[6] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0.025*size + posx, 0.05*size + posy);
            glVertex2f(0.05*size + posx, 0.05*size + posy);
            glEnd();
    }

    if(letter[7] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0 + posx, 0 + posy);
            glVertex2f(0.05*size + posx, 0 + posy);
            glEnd();
    }


//---------------slants /

    if(letter[8] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0.05*size + posx, 0.1*size + posy);
            glVertex2f(0.025*size + posx, 0.05*size + posy);
            glEnd();
    }

    if(letter[9] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0.025*size + posx, 0.05*size + posy);
            glVertex2f(0 + posx, 0 + posy);
            glEnd();
    }


//---------------slants \

    if(letter[10] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0 + posx, 0.1*size + posy);
            glVertex2f(0.025*size + posx, 0.05*size + posy);
            glEnd();
    }

    if(letter[11] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0.025*size + posx, 0.05*size + posy);
            glVertex2f(0.05*size + posx, 0 + posy);
            glEnd();
    }

    if(letter[12] == '1'){
        glBegin(GL_LINES);
            glVertex2f(0.025*size + posx, 0 + posy);
            glVertex2f(0.025*size + posx, 0.1*size + posy);
            glEnd();
    }




}

SegmentDisplay::SegmentDisplay(std::string letter){
    this->letter = letter;
}
```