Low Level Programing Post Mortem

In my option my project was overall successful as I did recreate a version of space invaders complete with win lose, main menu, restart and pause states as well as extras such as barriers that stop bullets but are destroyed over time. however, this does not mean the project is wit out its failings as the project relies heavily on static magic numbers that make the program less robust and easy to change also the collision system being used could do with improvements as currently all the necessary variables for Collison (size and position) are passed by reference in one by one where as a much more encapsulated system would possible put these variables into a bounding box vector and pass that across instead. Another small error with the game is that the ay out of the for loop for enemy firing and the probability check to make them fire at random times means that its more likely for the sprites on the top to fire rather than the ones on the bottom this is an error that can’t be fixed now but with more time could be fixed by restricting the code or possible adding a check to see what line just short if it is the same as the current shooting line then chose a new sprite to fire .If I had more time I would have include a feature where the game would shift if there was only one enemy left from space invaders to more of a Galaga type game where when the last sprite is killed a boss sprite move in fro off screen to the centre and start shooting out multiple bullets at the player and would have a health system so wouldn’t die in one shot (this would be similar to the barriers I implemented but would have been a child of enemy instead of game object) , I believe this would have made the game a lot more fun as it would have surprised the player and force them to use the skills they had learned in a new way .

OGLGame

InvadersGame

+InvadersGame();

+~InvadersGame();

+virtual bool run() override;

+bool shouldExit() const;

+void render();

+virtual bool init();

+virtual void drawFrame();

+void updateGame();

+void updateMenue();

+void updateGameOver();

+void updatePause();

+void updateReset();

+void updateWin\_Screen();

+void stateInput(int key, int action);

-void processGameActions();

-void input(int key, int action) ;

-int state\_callback\_id = -1;

-int callback\_id = -1;

-bool exit = false;

-int score;

-int ran;

-bool Spawned;

-const char\* scoreChar;

-std::string scoreString;

-std::unique\_ptr<Player> playerPt;

-std::unique\_ptr<Enemy> enemyPt;

-std::unique\_ptr<Bullet> bulletPt;

-std::unique\_ptr<EnemyBullet> enemybulletPt;

-std::unique\_ptr<Barrier> barrierPt;

-std::unique\_ptr<CollisionDetction> colisionPt;

-GameState game\_state = GameState::MAIN\_MENU;

GameObject

+GameObject() = default;

+~GameObject() = default;

+bool IsAlive();

-int Health;

Barrier

+Barrier();

+virtual bool init(std::shared\_ptr<ASGE::Renderer> renderer)

+float GetBarrierY(int i)

+float GetBarrierX(int i)

+float GetWidth(int i)

+float Gethight(int i)

+virtual void Render(std::shared\_ptr<ASGE::Renderer> renderer)

+void TakeDamge(int i)

+void deleteBarrier(int i)

+int getHealth(int i)

+void Reset()

-float Xpos

-float Ypos

-float Width

-float Height

-std::vector<bool>deadBarriers

-std::vector<int>health

-std::unique\_ptr<ASGE::Sprite> Barriers

-std::vector<std::unique\_ptr<ASGE::Sprite>>Blocks

CollisionDetection

+CollisionDetction()=default

+~CollisionDetction()=default

+bool hasThereBeenACollision( float AX, float AY, float AHeight,

float AWidth, float BX, float BY, float BHeight, float BWidth)

Actor

+Actor()=default;

+~Actor()=default;

+bool hasSpriteHitRightWall(int i, std::vector<std::unique\_ptr<ASGE::Sprite>>& ActorSprite);

bool hasSpriteHitLeftWall(int i, std::vector<std::unique\_ptr<ASGE::Sprite>>& ActorSprite);

void MoveRight(int i, std::vector<std::unique\_ptr<ASGE::Sprite>>& ActorSprite, int speed);

void MoveLeft(int i, +std::vector<std::unique\_ptr<ASGE::Sprite>>& ActorSprite, int speed);

-int speed;

Player

+Player();

+Player(Player&&rhs);

+virtual bool init(std::shared\_ptr<ASGE::Renderer> renderer);

+void Render(std::shared\_ptr<ASGE::Renderer> renderer);

+void moveLeft();

+void moveRight();

+virtual float GetXpostion(int i);

+virtual float GetYpostion(int i);

+float GetWidth(int i);

+float Gethight(int i);

+bool GetHasShot();

+bool SetHasShot(bool i);

+int playerHit();

+bool GetDeadSprites(int i);

+bool getIsAlive();

+void reset();

-bool hasShot = false;

-bool isAlive;

-int speed;

-int lives;

-float Xpos;

-float Ypos;

-float Width;

-float Height;

-std::unique\_ptr<ASGE::Sprite> player;

-std::vector<std::unique\_ptr<ASGE::Sprite>>Tank;

-std::unique\_ptr<ASGE::Sprite>DeadTankSprite;

-std::vector<bool>deadTanks;

Enemy

+Enemy();

+Enemy(Enemy&&rhs);

+bool init(std::shared\_ptr<ASGE::Renderer> renderer);

+virtual float GetXpostion(int i);

+virtual float GetYpostion(int i);

+float GetWidth(int i);

+float Gethight(int i);

+void Render(std::shared\_ptr<ASGE::Renderer> renderer);

+void Move();

+void MoveDown();

+bool GetDeadSprites(int i);

+void reset();

+bool areAllSpritesDead();

+bool hasEnemyhasWon();

+bool getHasEnemyWon();

+void killSprite(int i);

+void setEnemyWin(bool hasWon);

-std::unique\_ptr<ASGE::Sprite> Invaders;

-std::vector<std::unique\_ptr<ASGE::Sprite>>Enemys;

-std::vector<bool>Deadsprite;

-float Xpos;

-float Ypos;

-float Width;

-float Height;

-int speed;

-int howManyEnemiesKilled;

-int score;

-bool MovingRight;

-bool HasEnemyWon;

Bullet

+float GetBulletY();

+float GetBulletX();

+float GetWidth();

+float Gethight();

+virtual void Render(std::shared\_ptr<ASGE::Renderer> renderer);

+bool shoot(std::shared\_ptr<ASGE::Renderer> renderer,float xops);

+void MoveBullet();

+bool getBulletFierd();

+void setBulletFierd(bool hasBulletBeenFierd);

+bool deleteBullet();

-float Xpos;

-float Ypos;

-float Width;

-float Height;

-bool bulletFired;

-std::unique\_ptr<ASGE::Sprite> Bullets;

EnemyBullet

+float GetEnemyBulletY();

+float GetEnemyBulletX();

+float GetWidth();

+float Gethight();

+virtual void Render(std::shared\_ptr<ASGE::Renderer> renderer);

+bool shoot(std::shared\_ptr<ASGE::Renderer> renderer,float xops,float ypos);

+void MoveBullet();

+bool getBulletFierd();

+void setBulletFierd(bool hasBulletBeenFierd);

+bool deleteBullet();

-float Xpos;

-float Ypos;

-float Width;

-float Height;

-bool Fired;

-std::unique\_ptr<ASGE::Sprite> EBullet;