# Space Invaders – Class Breakdown

Actor

|  |
| --- |
| +enum ActorType{} |
| #unique\_ptr<ASGE::Sprite> avatar |
| #bool alive |
| #GameData\* game\_data |
| #int health |
| #int max\_health |
| #ActorType a\_type |

|  |
| --- |
| +ActorType getActorType() |
| +bool getAlive() |
| +void setAlive(bool a) |
| +int getHealth() |
| +virtual void damage() |
| +virtual void kill() |
| +void loadAvatar(\*renderer, filename, width, height) |
| +ASGE::Sprite getAvatar() |
| +bool Render(\*renderer) |
| +bool isCollidingWith(Actor\* other) |

Enemy : Actor

|  |
| --- |
| +enum EnemyState{ L,R,X } |
| -unique\_ptr<Bullet> e\_bull |
| -unique\_ptr<ASGE::Sprite> sprite\_l |
| -unique\_ptr<ASGE::Sprite> sprite\_r |
| -EnemyState state |
| -Explosion e\_explode |

|  |
| --- |
| +Bullet getBullet() |
| +EnemyState getState() |
| +void swapState() |

Player : Actor

|  |
| --- |
| +enum MovementState{ RIGHT, LEFT, NONE } |
| -unique\_ptr<Bullet> p\_bull |
| -float respawn\_timer |
| -float respawn\_max |
| -int lives |
| -MovementState state |

|  |
| --- |
| +Bullet getBullet() |
| +void tick() |
| +MovementState getState() |
| +void setState() |
| +int getLives() |
| +void kill() override |

Bullet : Actor

|  |
| --- |
| +enum BulletState{ L, R, X } |
| -unique\_ptr<ASGE::Sprite> sprite\_l |
| -unique\_ptr<ASGE::Sprite> sprite\_r |
| -Actor\* parent |
| -BulletState state |

|  |
| --- |
| +void shoot() |
| +void setParent() |
| +BulletState getBulletState() |
| +void swapBulletState() |
| +void updatePosition() |

Blockade : Actor

|  |
| --- |
| -unique\_ptr<ASGE::Sprite> sprite\_1 |
| -unique\_ptr<ASGE::Sprite> sprite\_2 |
| -unique\_ptr<ASGE::Sprite> sprite\_3 |

|  |
| --- |
| +void damage() override |

SoundSystem

|  |
| --- |
| +FMOD::System \*m\_pSystem |

|  |
| --- |
| +SoundSystem() |
| +void createSound() |
| +void playSound() |
| +void releaseSound() |

Explosion : Actor

|  |
| --- |
| -unique\_ptr<ASGE::Sprite> sprite\_1 |
| -unique\_ptr<ASGE::Sprite> sprite\_2 |
| -Actor\* parent |
| -float tick\_timer |
| -float tick\_max |
| -bool second\_sprite |

|  |
| --- |
| +void damage() override |
| +void tick() |
| +void detonate() |
| +void reset() |
| +void setParent() |

enum class GameState

|  |
| --- |
| MENU |
| GAMEPLAY |
| GAMEOVER |
| GAMEWIN |
| GAMELOSE |
| PAUSE |

EnemyManager

|  |
| --- |
| +enum OtherObjType{ BLOCKADE, BULLET, PLAYER } |
| -vector<unique\_ptr<Enemy>> enemies |
| -unique\_ptr<Enemy> ufo |
| -const int num\_rows |
| -const int num\_cols |
| -const int num\_enemies |
| -float enemy\_tick\_timer |
| -float enemy\_tick\_max |
| -float bullet\_tick\_timer |
| -const float\_bullet\_tick\_max |
| -float speed |
| -const int ufo\_probability |
| -bool moving\_right |
| -GameData\* game\_data |

|  |
| --- |
| +void tick() |
| +int getEnemiesAlive() |
| +void flipEnemies() |
| +void moveEnemies() |
| +void moveUFO() |
| +void renderEnemies(\*renderer) |
| +void updateSpeed() |
| +void checkCollisionsWith(Actor\* other, OtherObjType type) |
| +void checkBulletCollisionsWith(Actor\* other, OtherObjType type) |
| +void increaseDifficulty() |

Game

|  |
| --- |
| -unique\_ptr<GameData > game\_data |
| -unique\_ptr<ASGE::Sprite> sprite |
| -unique\_ptr<ASGE::Sprite> pause\_screen |
| -unique\_ptr<ASGE::Sprite> gameover\_screen |
| -unique\_ptr<Player> player\_one |
| -unique\_ptr<EnemyManager> enemy\_man |
| -bool exit |
| -vector<unique\_ptr<Blockade>> blockades |
| -const int num\_blockades |
| -int callback\_id |

|  |
| --- |
| +virtual bool run() override |
| +bool shouldExit() const |
| +void render() |
| +virtual bool init() |
| +virtual void drawFrame() |
| +virtual bool initGamePlay() |
| +virtual void drawFrameGame() |
| +virtual void checkCollisions() |
| +virtual void resetLevel() |
| +virtual bool initPause() |
| +virtual void drawFramePause() |
| +virtual bool initGameover() |
| +virtual void drawFrameGameover() |
| +void processGameActions() |
| +void input(int key, int action) |
| +void initBlockades() |
| +initSoundSystem() |

enum class GameAction

|  |
| --- |
| INVALID |
| NONE |
| EXIT |
| PAUSE |
| MOVERIGHT |
| MOVELEFT |
| SHOOT |
| MENUSELECT |
| MENUUP |
| MENUDOWN |

GameData

|  |
| --- |
| +GameState state |
| +int score |
| +float difficulty\_divisor |
| +double delta\_time |
| +std::shared\_ptr<ASGE::Renderer> renderer |
| +FMOD::System sound\_sys |

# Class Functionality

* Actor – A basic game object with a sprite, a health value, functions for loading, damaging and killing, and a collision function that takes another actor as a parameter.
  + Player – The laser cannon at the bottom of the screen. Owns a bullet, and can move left and right. Explodes when hit, losing a life, and respawns (provided the lives counter has not dropped to zero).
  + Enemy – The aliens that advance on the player from above. They have two frames of animation each, as well as a bullet. They explode when hit but do not respawn until the level is reset. The UFO is also an enemy, but it is ticked separately as the enemy manager makes it behave differently.
  + Bullet – A projectile that travels vertically, independent of its parent. They can have up to two frames of animation, and have different travel directions and speeds depending on if they were shot from a player or an enemy. I made the two kinds of bullet into one because much of the functionality is shared between them.
  + Blockade – These are the ‘bunkers’ that help protect the player from enemy fire. They are each made up of six segments that, when shot by either the player or an enemy, will weaken and eventually break. I chose six segments because I am not confident in my ability to create a destructible pixel grid like in the original space invaders, but I did want there to be individual pieces to capture the feeling of the original.
  + Explosion – An actor similar to the bullet but with no movement. They have two frames of animation, and automatically shut themselves off after a set period of time. I made this its own class rather than a part of another so I could reduce code duplication between destructible objects. By passing in different parameters, I could create different kinds of explosion for different objects in the future.
* EnemyManager – This object, owned by the game, handles all the logic relating to the aliens. This includes their speed, direction of travel, synchronising their animations to their movements, their rendering, and their collisions. I decided to use a manager for the enemies because having 50+ enemies in the Game class would bloat it and would soon become cumbersome. By separating them out to their own manager, I was able to significantly cut down the number of lines of code in Game, as well as encapsulate the enemies.
* GameData – This data-only object holds all the useful variables to be passed around to multiple objects. The game state, number of lives, score, difficulty, delta time, sounds and a pointer to the renderer are all stored here. I made this object so every other class didn’t need ten construction parameters. By keeping all the data centralised, objects can indirectly communicate with each other in a non-obstructive way.
* Game – The main game loop that initialises everything, delegates ticking and drawing to the appropriate managers, checks for collisions and manages the switching of game states.
* GameAction – An enum class defining the different actions the player can take. This makes the ‘processGameActions’ function easier to read.
* GameState – An enum class defining the different states (such as menu, gameplay, paused etc) the game can be in. This allows me to easily swap between gameplay states without too many ‘if’ statements.
* SoundSystem – A wrapper for an FMOD sound system, that simplifies certain actions. It is used to handle all the loading and playing of sounds. I used this to replace the Windows built-in sound system because FMOD supports multiple tracks, allowing multiple sounds at once. This is beneficial to the UFO, as it has a looping sound that would otherwise get interrupted by other sounds.