# Side Scrolling Shmup Design Document

### To talk abt

Ship feel

Ent system

Ent store system

Ship obj

Shooting cooldown

No ent create or destory during gameplay (all pooled)

Ship spawn invuln

Pitfalls of a pre collision check on spawn?

To be inproved:

Input buffer

Balence

Theming – why asteroids when not in space

Animated sprites – i might act do this one (bullet and asteroids)

Stretch and squash ship with velocity/accell

TODO:

Bullet cooldown post mortem

Psudocode??!?!

Lifetime respawn anticollision system

Where ents are stored

How textures are stored and loaded

Concessions for time/complexity

iFrames on start

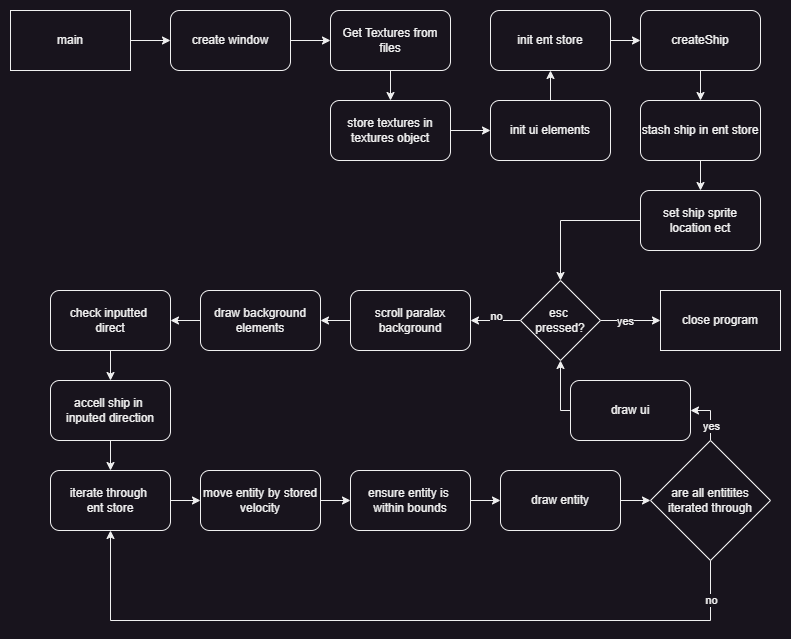
### Understanding the template

Due already having implemented a full entity system i will be using the my homework 3 solution as a jumping off point, rather then solution 11.

#### Procedure Dependency Diagram



#### Flowchart



## Upgrading the template

Alot of the ground work for this project is already present thanks to my work on homework 3, a robust entity system and entity store to easily manipulate a large number of entities.

In order to meet to specification i plan to extend the entity class to include collision as well as logic for collision cases. I plan to make a pool of both asteroids and bullets, in order to handle the large number of entities in an efficient manner, managing them with the entity store. This ship must be able to shoot, with the life of the bullets and asteroids handled correctly, respawning and activating/deactivating pooled entities correctly. The game must also have a fail state if a ship collides with an asteroid.

### Collision detection and anti collision

When testing the ideas to avoid overlapping asteroids, i was experiencing difficult to debug issues regarding collision detection based on predicting asteroid location, due to ensuring that the asteroids are no half off screen, i realised that by simply storing the lifetime of the entity (as i was planning on doing anyway) i could use that to move newly spawned asteroids when they, rather then relying on the prediction, as i was going to have to implement this solution for the bullet system later anyways. As such we do not need to run this prediction in the planned program.

E.g.

|  |
| --- |
| onAsteroidSpawn: if ((lifetime < SOMECONST) AND (ent.isColliding)) then {respawn} |

So collision function will look something like this

|  |
| --- |
| Foreach (entity : i):  if ((i.active) AND (i != currentEnt) AND (i.isColliding){  if ((i.type == asteroid) AND (i.lifetime < i.LIFETIMERESPAWNLIMIT)){  i.respawn()  }  if ((i.type == bullet) AND (currentEnt.type == asteroid)){  currentEnt.respawn()  i.deactivate()  }  if ((i.type == asteroid) AND (currentEnt.type == ship))  entStore.deactivateAllEnts()  deathMessage.activate()  }  }  } |

### Pooling entities

In order to make the game more preformant and the entity pointers more manageable all entities are going to be created at the start of the game, then simply manipulated in order to give the illusion that they are being created and destroyed. For the asteroids this is simple enough as they just need to be moved from one side of the screen to the other when they leave vision of the player. However for bullets we will require activating and deactivating them as required from the pool, to avoid them colliding with asteroids when deactive. All entities in the project are stored within the entStore object.

### Fail State

Due to how simple the gamestate is when this ship collides with a i plan on deactivating all entities and showing a simple text element over the screen.

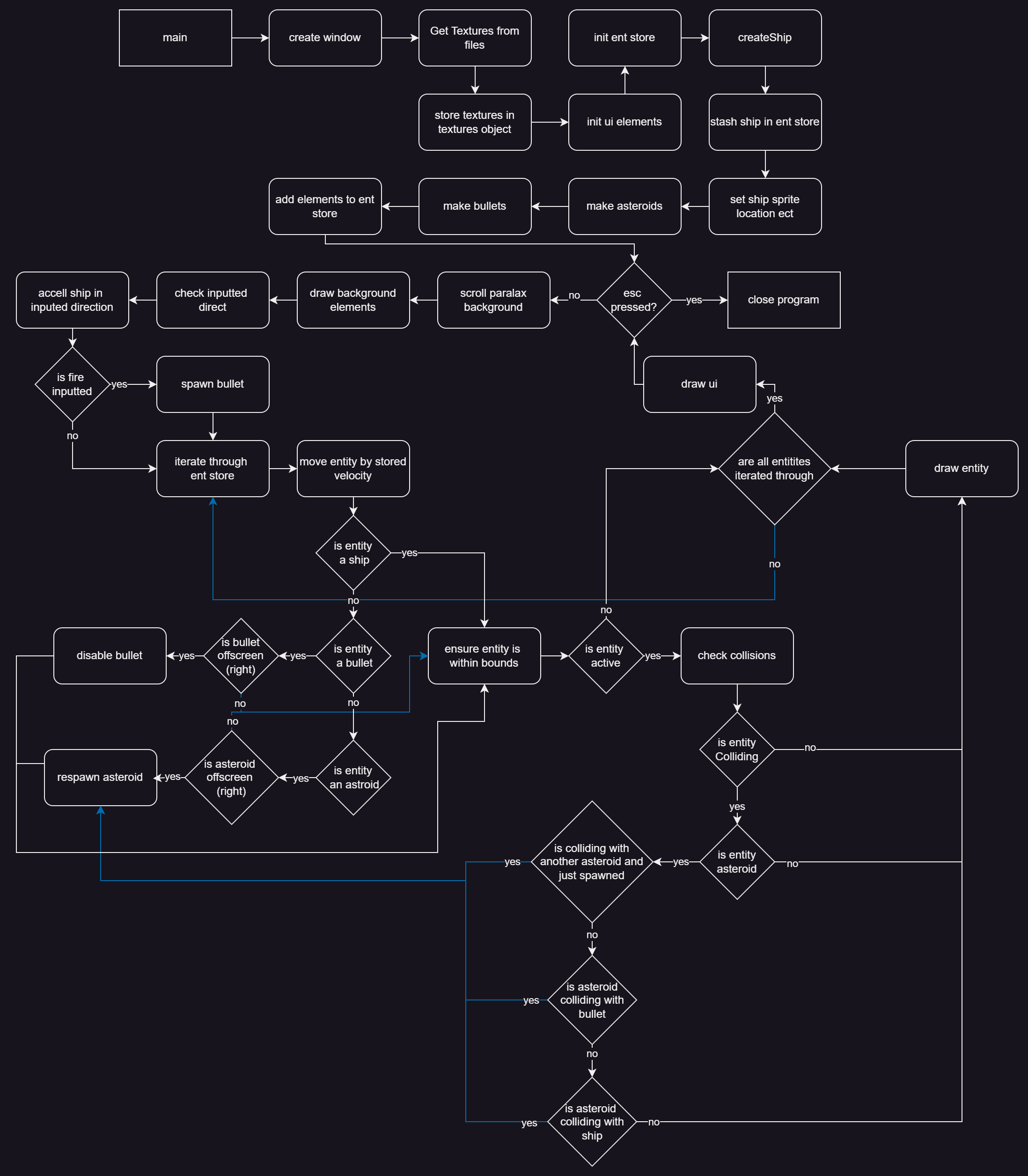
### Storing Textures

In order to allow for a easy, performant and extendable texture management system i would like to store each texture only once (for performance after the game is loaded), be able to easily find these textures in memory in a human readable manner, and have a single simple place in the code to add or remove these textures from the project.

### Animation

TODO

### Upgrade plan flowchart



### Upgrade plan procedure dependency diagram

# Postmortem