# Initial Design

I will be modelling my system on atmospheric flight. The program will be a side-scrolling computer game where the player is in a tunnel moving right where they have to dodge obstacles that get in their way. Picking up items on the way will reward the player with boosts and power ups.

**Memory allocation:**
The game will be split up into different objects that perform different tasks. The following elements will have their own objects:  roof and floor of the tunnel will be a collection of border-specific objects, the players sprite, the obstacles, the refresher that prints the frames to the screen, and the high scores list. Within each of these objects there will be variables and arrays, that interact with each other.

**Memory management:**
An example of how I will deal with memory leaks is when an obstacle object is instantiated, it will slide across the screen and when it leaves there is no use for it any more, so it will be deleted. When the borders of the game are refreshed, they will be copying the contents of the arrays holding the information, rather than copying pointers.

**Arrays:**
The majority of this program will be array manipulation. To create the “movement” of the tunnel I will be using a circular array that simulates an array shuffling from right to left, but in reality the contents won't move at all, the new element appearing on the right hand side will just overwrite the element disappearing on the left

**Strings:**
When the border arrays shuffle from right to left, the characters to be replaced will come from a string. A loop will iterate through the strings characters, copying them them into the the border array.

**Numerical output:**
Integers will be displayed in many different ways. There will be a counter displaying the current progression in meters, the high score will be displayed on the screen, and there will be an option to look at the high scores.

**User Input:**
The game will take a variety of different user inputs. When the game starts, there will be a menu screen prompting the user to either start the game, look at the high scores page, or quit. Hitting “spacebar”, “q”, or “h” will trigger one of these three events. When the game is running, the player will press spacebar rise their character sprite. When the game ends, The user will be prompted to enter their name, which will be then saved in a high scores file.

**Inheritance:**
All obstacles will be structured in a multi-levelled inheritance hierarchy tree. The base class will be an ordinary object that comes towards you. Two classes will inherit from this class: a power up class, and a dynamic obstacle class. From the power up class there will be two other classes that inherit from it: a speed up boost, and an invulnerability class. From the dynamic obstacle class there will be two classes that inherit from it: an obstacle that moves up/down, and an obstacle that moves left/right.

**Polymorphism:**
All objects whose class stem from the superclass “Obstacle” will all have the method “interact()” which will be called when the player collides with the obstacle. Depending on which child class has been activated, the interact function will do different things. If a power up object collides with the player, the game will not end, and some event will be triggered. If a “bad” obstacle collides with the player, the game will end. Every class that inherits from the obstacle superclass will all receive different instructions depending on which class it is.

**OO Programming and Design:**
In the game, every different element will have its own class whose object will interact with other objects in a specific way. If an object needs to have its data manipulated, it will store that method within it self. If an object needs to manipulate another object, it will call a method within that class, requiring no additional code. The hierarchy of obstacles will each contain a branch that is very different from its parent and will perform an important roll, nothing will be implemented without a proper use.

**Testing**
Every step in the program creation will follow with numerous tests to make sure things are working correctly. I will not move forward in the program design unless the task I am currently working successfully compiles and works as intended. The testing is one of the most important phases, as it helps me understand my code better, aiding my coding skills.

**Plan**
The computer game will be rendered in terminal the using ncurses library. Each horizontal line on the display will be an array of characters. Every time the game refreshes, each array will shuffle its contents along from right to left, and then a print function will be called to make ncurses print the next frame to the screen. The roof and floor of the screen will be random capitalised words related to the game that will move from right to left. Each horizontal character array for the roof/floor will be an instance of a single class that has a methods to choose a new random word to shuffle along. This enables each line on the screen to print words independently of the other rows.

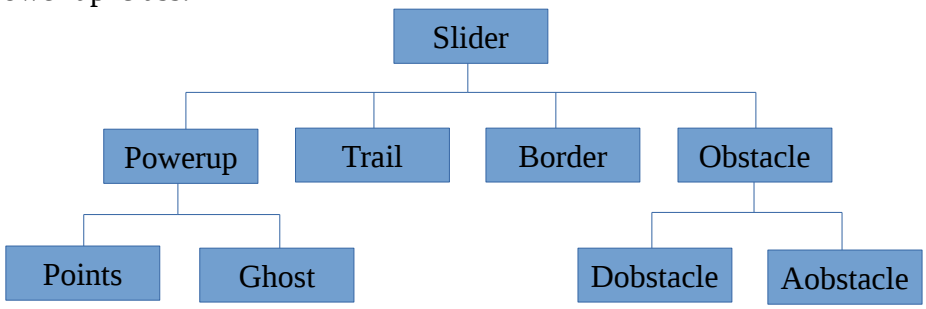
Every few seconds an obstacle will emerge from the right hand side of the screen and move left at a constant rate. The obstacle will look and behave differently depending on what class has been instantiated. All obstacles will share the traits of moving right to left, however some can help you and some can hurt you. Which obstacles get instantiated will depend on a random number generator. In order to get the object to move along the screen, every time the screen refreshes it will call a method of the objects class that moves its coordinates, allowing it to be reprinted in a different location.

The player's character sprite will stay near the left hand side of the screen, not being able to move horizontally, but will be able to move vertically. In order to achieve this, the boundaries and obstacles will update first, and if the players sprite gets copied on top of another object, the object will call an “interact” method. Depending on what object the interact method gets called to, different things will happen. If the object is a power up, the icon for the power up will be removed from the screen, and the effect will occur. If the object is an obstacle or a border, the “interact” function will stop the main game loop, ending the game.

At the bottom left of the screen there will be an increasing counter displaying the players current score, and on the bottom right there will be the players high score. When the player “dies”, ending the game, a screen will appear prompting the player to enter their name. When the player enters some text, that text will be stored in a text file along with the score of that current game run. This file will be the high scores file that can be called from the start menu to be viewed.

**Heirachy**

In my design, everything that moves left will inherit from the super class 'Slider'. All the obstacles that kill the player will be under the 'obstacle' class, and all obstacles that help the player will be under the 'power up' class.




**Object Relationship**

The main function has a loop that calls the game loop from the game class. The controller is initialised by main, but shares a global variable with the game loop. The game loop instantiates and deletes all obstacles and the player. The main function initialises the borders to begin with, but passes control to game loop. 'Border' has control of 'Textblock', which has control of 'Textline'.