# **Arkanoid : development notes**

## Engine

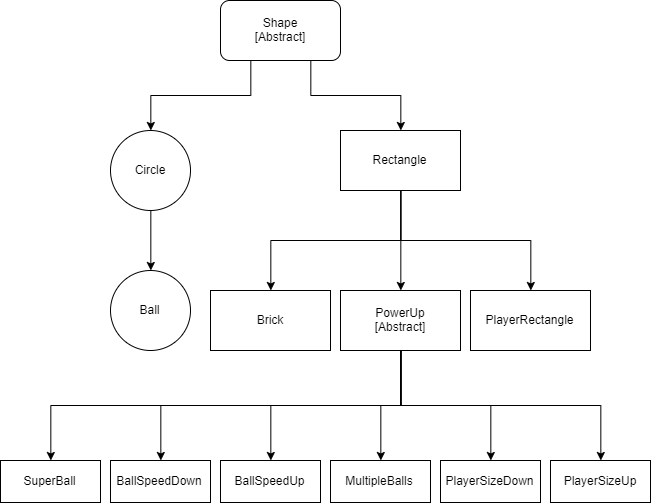
I chose SFML to create a small Arkanoid like game. It is a simple library that gives me access to useful things. When you start the solution, the engine is created, and then run continuously if the window is open.

The engine does 3 main task every tick:

* ProcessInputs (gets the input from the user)
* Update (movement and collisions)
* Draw (rendering)

I did not follow the exact game design of the original game, but I did not diverge too much either. My programming style is straightforward. I do not use complicated or fancy logic. I prefer to use simple and organized code so if someone else must work or read the code, he can do so without me having to provide explanation.

## Inheritance Graph



## Design patterns

## Factory method

I choose to use a factory method to create the power ups and rely on inheritance. The base class Power up implements a static “Create” method that accepts a power up type. This method then returns a shared ptr of the corresponding derived class.

Looking back, I think one power up class would have been enough for a simple game with a finished scope. However, I made this choice because I did not know how simple they were at first and because if the game becomes bigger it will be easier to add more complex power ups.

## Singleton

I used a singleton for the “Engine” class. There will only be one engine only and the game is very simple, I found that the singleton would not be too bad in this case. I wanted to get the game as quickly as I could, and the singleton is very useful for that. However, I would usually not use a Singleton and prefer Events and/or delegates.

## Update

As I have worked with mostly Unreal engine and Unity so far, I decided to try to replicate their core. Therefore, each entity has an Update method that will be called from the Engine main loop. If I had to make the project bigger or create a full engine, I would have the engine have a list of “Updatable” which is either the first parent class (above parent) or an interface, and update all his “Updatable” in his Update loop. This way, it would be more like Unity and Unreal.

## Miscellaneous

## Brick set up

To set up the bricks, I decided to read information from the pixel of a Texture. The rule is the following:

* The pixel index is used as the offset to place the brick
* The colour of the pixel is used for the colour of the brick
* The opacity of the pixel is used to determine the HP of the brick (10% opacity = 1 HP)

## Tweakable Variables

For simplicity and development, I left the tweakable variables in #define in the “Engine” header file. Those should have limit to prevent a person to put values that could break the game (clamp). I put some safeguards in the code, but if those variables would be exposed to designers, they should be exposed in a way that they are clamped to valid values. Also, the best would be to extract the data from the code and have them in an external file.

## Shared Texture

In SFML, loading multiple textures is not recommended. Therefore, I used one single texture for all the sprites of the game. The same should be done for level templates.

## Collision handling

The collision is handled by the Shape class that has a virtual OnCollision(pointer \_other shape) method. I rely on polymorphism to have each derived class override this method and have their own custom behaviour. By passing a pointer to the other shape the derived then can cast that pointer into a pointer of any other derived type. This prevents to have to check specific collisions between different shapes.

## Improvements

I still would like to make the 2 other power ups: sticky ball and shooting bullets from the player. I also would like to make more crazy power ups.

I should also improve my brick set up process so it could read any texture (with a transparent or white background) and create level from those. In that way, I could just go on google and take cool pixel art images, drag them into the project and voila!

I should also improve the collision system, which is very basic and sometimes has some hick ups.