Assignment 2

30 september 2016 - Group 666

Exercise 1 - Design patterns

1. Why and how

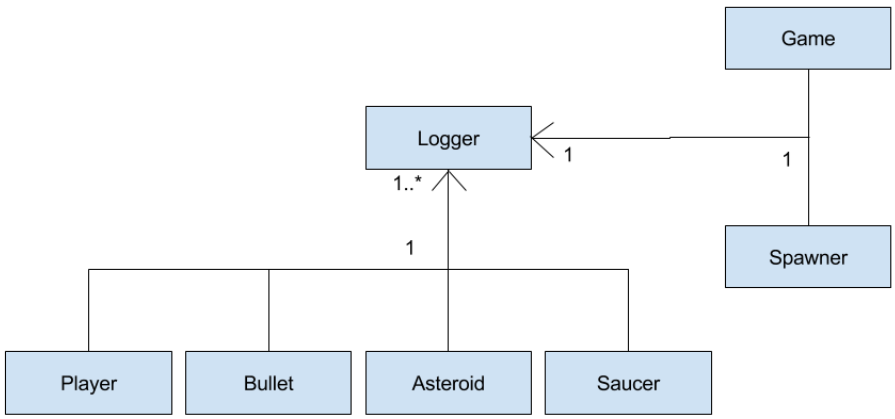
The 2 chosen design patterns: Singleton, Composite.

Singleton:
For our logger we use the singleton design pattern. This is because we want all classes to use the same logger. We ensure thread-safety through creating a static instance and having a method for other classes to access this instance.

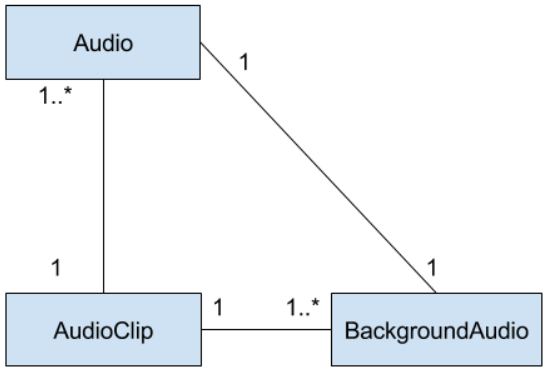
Composite:
Our Audio class uses a composite structure. It links to a variety of AudioClips outside classes can operate on, but it also links to another class similar to it to delegate certain functions. Via the audio class, the various AudioClips can be accessed in a way similar to how the Audio class itself is accessed.

2. Class diagram

Singleton:

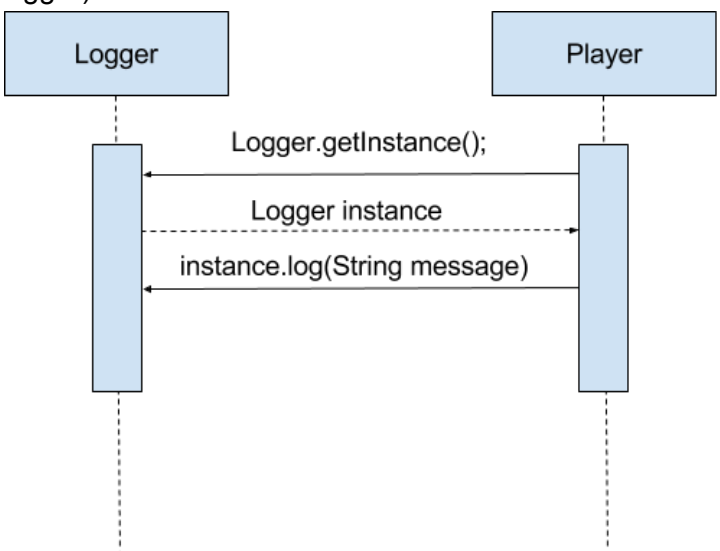


Composite:

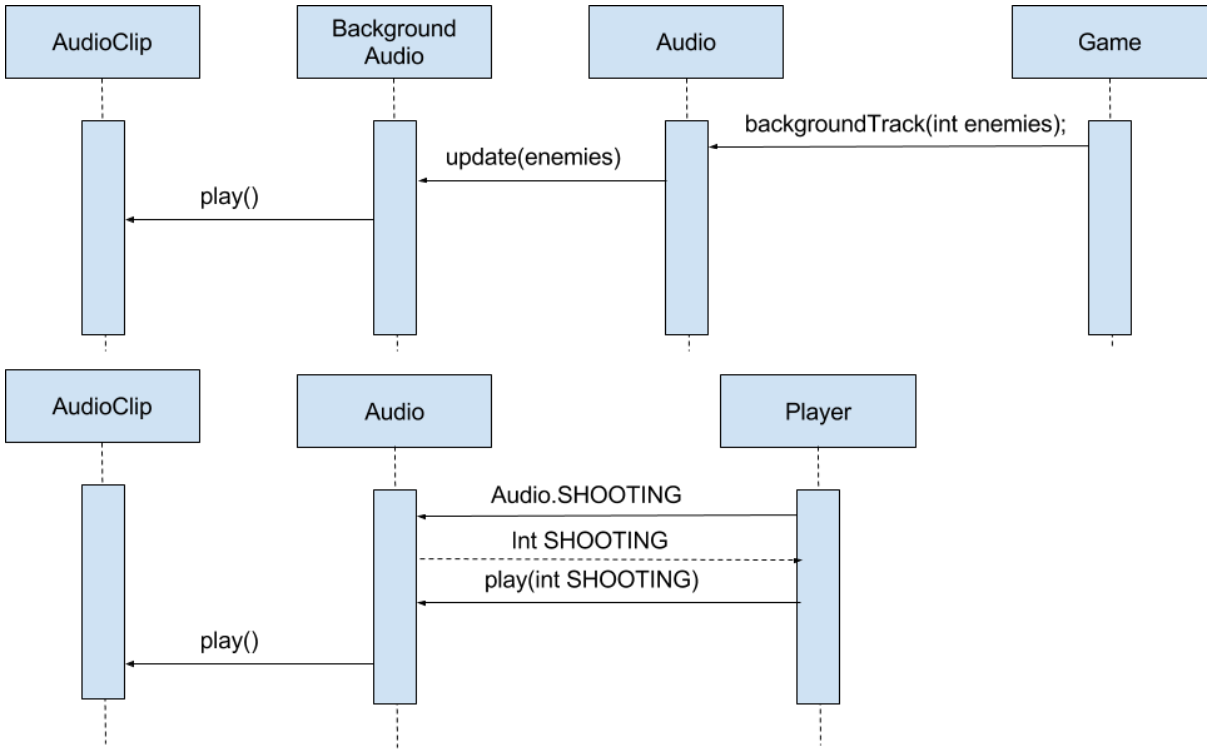


3. Sequence diagram

Singleton (Player class is used as example, but this goes for any class which uses the logger):



Composite:



Exercise 2 - Your wish is my command

1. Requirements TA

Local Multiplayer

Must haves:

1. The game shall have the gamestate CoOp.
2. When the game is in the gamestate CoOp the game shall spawn two players: player 1 and player 2.
3. The two players shall be controlled independently on the same keyboard.
4. Player two shall have separate lives from player one.
5. Both players shall get an extra life every 10k points.
6. Player two shall have a different appearance from player one.

Should haves:

1. Any player shall be able to revive the other player when this player has no more lives left by getting another 10k points.

Could haves:

1. Player one shall be able to hit player two (Friendly fire).
2. CoOp shall have its high score saved separately from single-player.

Won't haves:

1. Players shall have separate score (so they will not have this! Otherwise it wouldn't be CoOp.)

2. RDD & UML

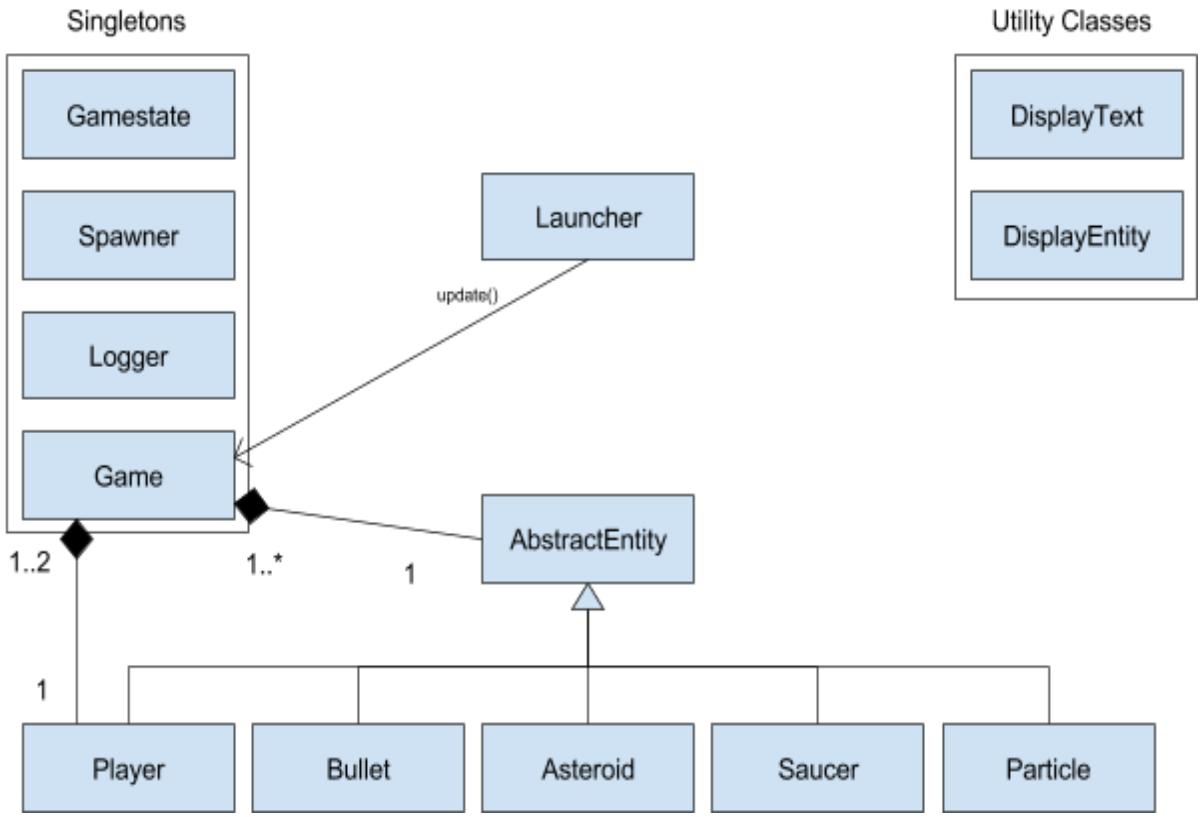
Obvious classes	Uncertain candidates	Nonsense
Game	Gamestate	1, 2 (player)
Player	lives	keyboard
	points	CoOp
	appearance	

Game and Player already are classes. Gamestate lives points and appearance are attributes. There will not be any new classes. But players will get different appearances. The crc only has an updated card:

Player	
Superclass(es):	
Subclasses:	
Accelerate, etc (old)	Entity
Have appearance	Player

Comparison

The requirements did not indicate any big scale changes. But during the implementation a couple of big changes have emerged. First of all, Gamestate and DisplayEntity are new classes and Display is renamed to DisplayText. Gamestate regulates the game states, this was previously done in the Game class. The DisplayEntity now holds all draw() methods of the entities. Secondly, Gamestate, Spawner and Game are now also singletons, just like Logger. Now only the game has a couple of other classes, the rest uses the getInstance() methods of the singletons. At last, the Game class can now have two players instead of just one. This is needed for multiplayer.



Exercise 3 - 20-Time

1. Requirements

Functional requirements

Must haves:

- Powerups shall spawn regularly.
- Powerups shall modify some aspect of the game if the player flies into them.
- If the player flies into a powerup, the powerup entity will disappear.
- Powerups shall have a set lifetime, after which their effects will wear off.
- Powerup entities shall be visible on the game “board”.
- One type of powerup shall grant the player an extra life.
- One type of powerup shall grant the player a shield.
- A shield shall take one hit for the player, preventing the player from losing a life.
- One type of powerup shall increase the size of the player’s bullets.
- One type of powerup shall grant the player the ability to fire piercing bullets.
- Piercing bullets can go through multiple asteroids.
- One type of powerup shall increase in the number of bullets a player can have on-screen at the time and the rate at which with they fire.
- One type of powerup shall grant the player the ability to fire three bullets at once (with a corresponding 3x increase to the number of bullets the player can have on-screen)

Should haves:

- The remaining lifetime shall be visible to the player in an explicit manner.
- Different powerups shall have a different icon on the game “board”.

Could haves:

- Having a powerup shall cause an icon corresponding to the type of powerup to be displayed next to the player’s lives.
- There shall be the option to turn powerups on/off.
- Only one shield shall be usable at a time.

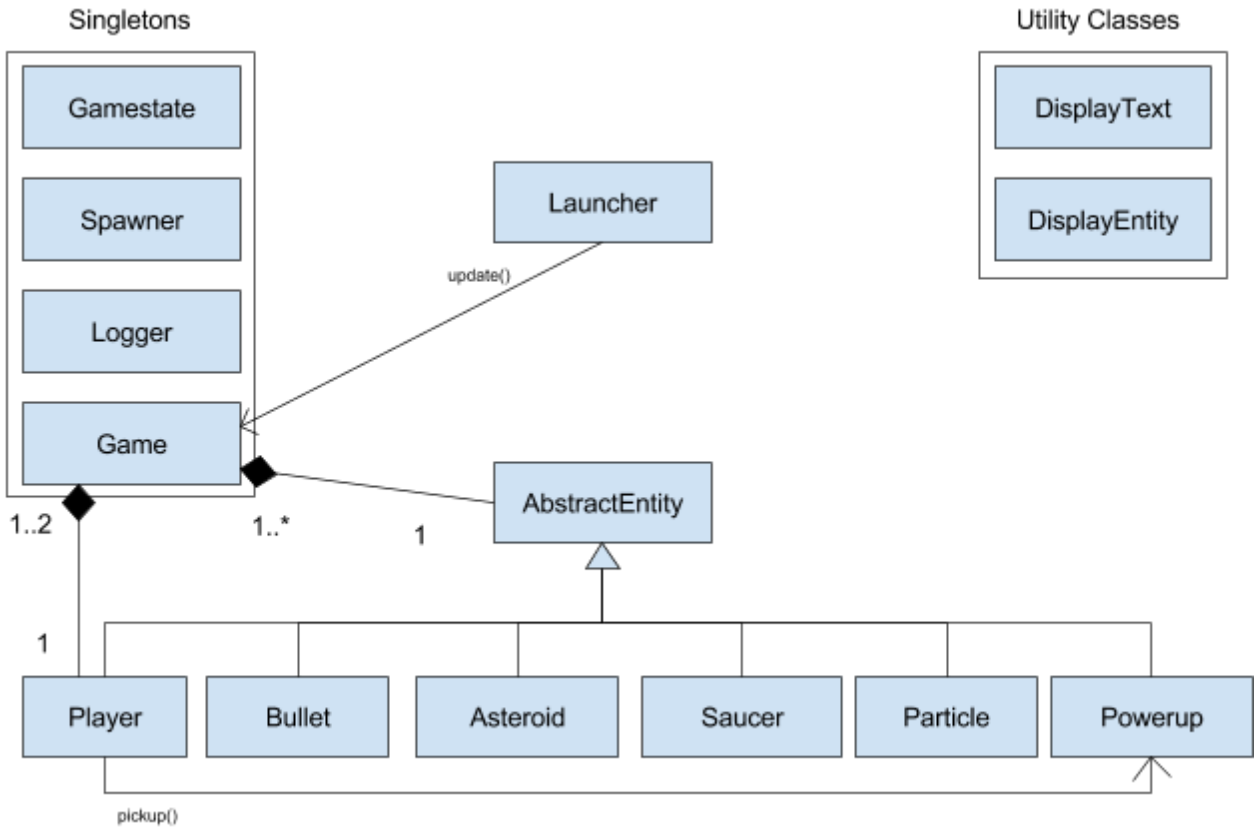
2. RDD & UML

Obvious classes	Uncertain candidates	Nonsense
Powerup	lives	
Bullet	powerup	
Player	shielding	
Spawner	piercing	
	tripleShot	
	bulletSize	
	POWERUP_TIME	
	startPowerupTime	

Bullet and Player already are classes. Lives are an attribute. There will be one new class, Powerup. Bullets will need to have a new attribute, piercing, which will determine how many asteroids it can destroy. By default this will be set to one, and decreased each time it hits an asteroid. Bullets are destroyed if they have penetration power 0. Player needs new setting methods as well to allow powerups to affect the player's abilities. Furthermore, to store the result of certain powerups, it will have additional attributes shielding (stores the number of shields the player has left), piercing (stores how far bullets should be able to penetrate a target) and tripleShot (which stores whether the player can fire three bullets or not), along with bulletSize (stores the size bullets fired by the player should be). To fire three bullets at once, Player needs an extra method that implements this. To cause powerups to be spawned, Spawner needs to be modified as well, to include an attribute that keeps track of the time between powerups (called and a method to actually spawn them.

Comparison

Changes had to be made among several classes to accommodate the new class. However, in terms of UML, it made very little difference, as most of the changes involved the interaction of classes that already had some form of interaction. Powerup is a new addition to the UML scheme, and its pickup() method allows Player to get some information from it.


Singletons
Utility Classes
Gamestate
Spawner
Logger
Game
DisplayText
DisplayEntity
Launcher
update()
AbstractEntity
1..2
1..*
1
1
Player
Bullet
Asteroid
Saucer
Particle
Powerup
pickup()