Object Oriented Analysis and Design Project 7: Kitchen Cleanup

Samuel Mast, Benny Sakiewicz, and Liam Semeria

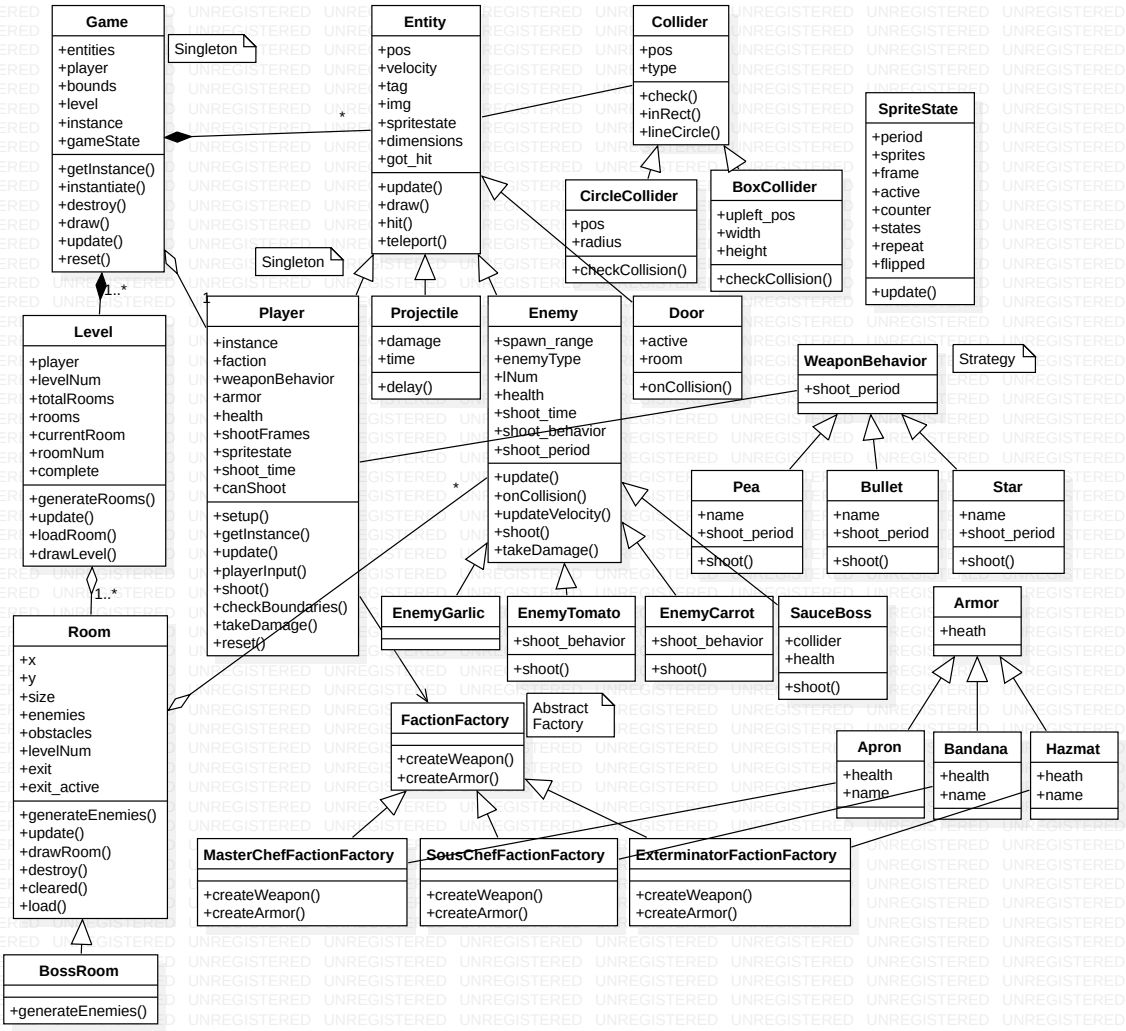
Spring 2022

State of System

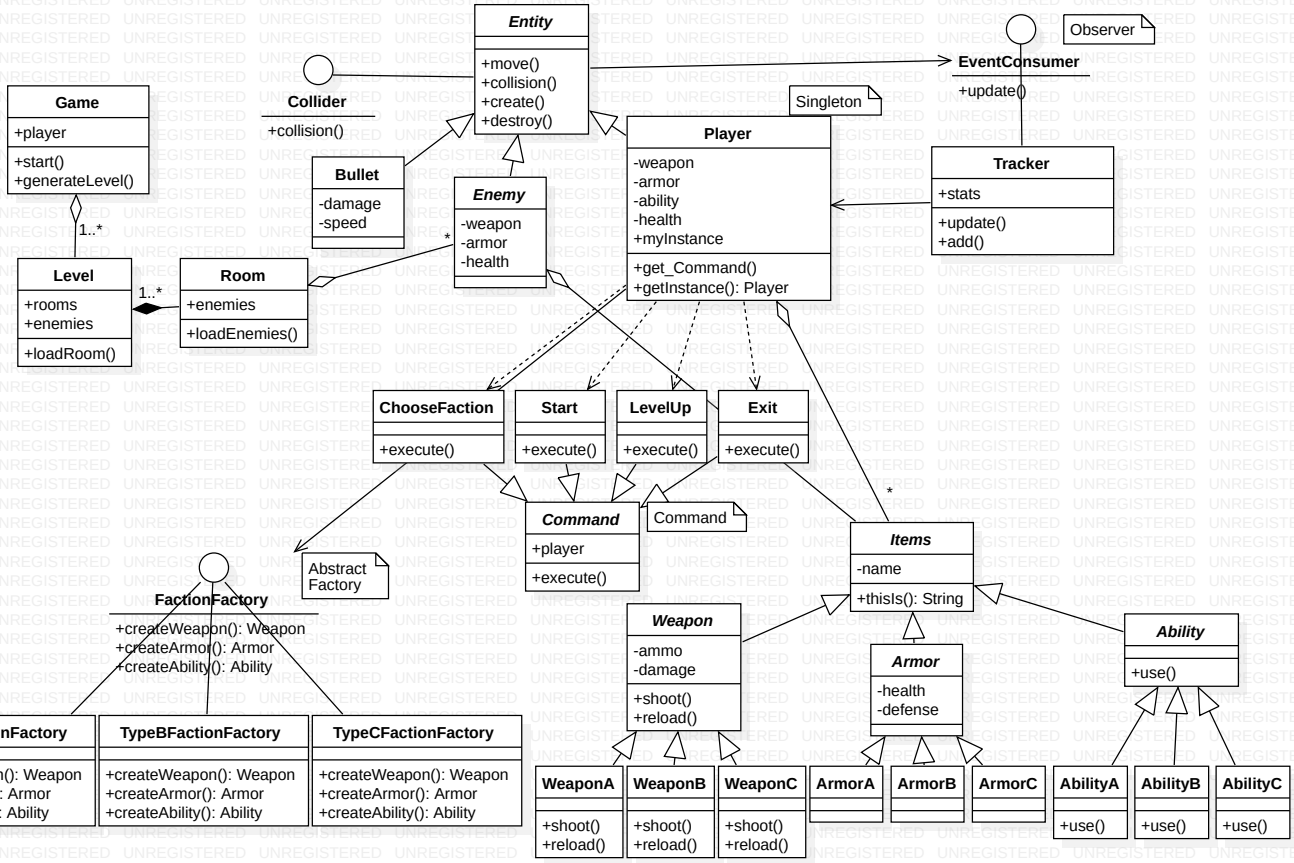
The final state of our system is a completed game product. We were able to implement a controllable player with three different factions that the user can choose from. Each of these has different weapons and armor. We implemented multiple different enemy types with unique attack patterns for each. We were able to implement a main menu where the user starts and selects their faction and then returns to when they die. We also implemented a level system where once the player beats all of the randomly generated enemies in a room they are able to go to a portal that takes them to the next level. We did not end up implementing abilities or picking up different items as we wanting to focus on making the core game play very well made and those ending up being out of scope as they were already stretch goals to begin with.

Class Diagram and Comparison

Final Class Diagram:


Game
+entities
+player
+bounds
+level
+instance
+gameState
+getInstance()
+instantiate()
+destroy()
+draw()
+update()
+reset()
Singleton
Entity
+pos
+velocity
+tag
+img
+spritestate
+dimensions
+got_hit
+update()
+draw()
+hit()
+teleport()
Collider
+pos
+type
+check()
+inRect()
+lineCircle()
SpriteState
+period
+sprites
+frame
+active
+counter
+states
+repeat
+flipped
+update()
CircleCollider
+pos
+radius
+checkCollision()
BoxCollider
+upleft_pos
+width
+height
+checkCollision()
Singleton
Player
+instance
+faction
+weaponBehavior
+armor
+health
+shootFrames
+spritestate
+shoot_time
+canShoot
+setup()
+getInstance()
+update()
+playerInput()
+shoot()
+checkBoundaries()
+takeDamage()
+reset()
Projectile
+damage
+time
+delay()
Enemy
+spawn_range
+enemyType
+INum
+health
+shoot_time
+shoot_behavior
+shoot_period
+update()
+onCollision()
+updateVelocity()
+shoot()
+takeDamage()
Door
+active
+room
+onCollision()
Level
+player
+levelNum
+totalRooms
+rooms
+currentRoom
+roomNum
+complete
+generateRooms()
+update()
+loadRoom()
+drawLevel()
WeaponBehavior
+shoot_period
Strategy
Pea
+name
+shoot_period
+shoot()
Bullet
+name
+shoot_period
+shoot()
Star
+name
+shoot_period
+shoot()
Armor
+heath
Room
+x
+y
+size
+enemies
+obstacles
+levelNum
+exit
+exit_active
+generateEnemies()
+update()
+drawRoom()
+destroy()
+cleared()
+load()
EnemyGarlic
EnemyTomato
+shoot_behavior
+shoot()
EnemyCarrot
+shoot_behavior
+shoot()
SauceBoss
+collider
+health
+shoot()
FactionFactory
+createWeapon()
+createArmor()
Abstract Factory
Apron
+health
+name
Bandana
+health
+name
Hazmat
+heath
+name
MasterChefFactionFactory
+createWeapon()
+createArmor()
SousChefFactionFactory
+createWeapon()
+createArmor()
ExterminatorFactionFactory
+createWeapon()
+createArmor()
BossRoom
+generateEnemies()
1..*
1
*
1..*
*

Project 5 Class Diagram:


Entity
+move()
+collision()
+create()
+destroy()
Collider
+collision()
Observer
EventConsumer
+update()
Game
+player
+start()
+generateLevel()
Singleton
Player
-weapon
-armor
-ability
-health
+myInstance
+get_Command()
+getInstance(): Player
Tracker
+stats
+update()
+add()
Bullet
-damage
-speed
Enemy
-weapon
-armor
-health
1..*
Level
+rooms
+enemies
+loadRoom()
1..*
Room
+enemies
+loadEnemies()
ChooseFaction
+execute()
Start
+execute()
LevelUp
+execute()
Exit
+execute()
Command
+player
+execute()
Command
Items
-name
+thisIs(): String
Abstract Factory
FactionFactory
+createWeapon(): Weapon
+createArmor(): Armor
+createAbility(): Ability
Weapon
-ammo
-damage
+shoot()
+reload()
Armor
-health
-defense
Ability
+use()
TypeAFactionFactory
+createWeapon(): Weapon
+createArmor(): Armor
+createAbility(): Ability
TypeBFactionFactory
+createWeapon(): Weapon
+createArmor(): Armor
+createAbility(): Ability
TypeCFactionFactory
+createWeapon(): Weapon
+createArmor(): Armor
+createAbility(): Ability
WeaponA
+shoot()
+reload()
WeaponB
+shoot()
+reload()
WeaponC
+shoot()
+reload()
ArmorA
ArmorB
ArmorC
AbilityA
+use()
AbilityB
+use()
AbilityC
+use()

Key Changes:

One of the most major changes from our initial idea was not using command for a few different tasks. We find that while actually implementing it most of the tasks that we thought might initially need some sort of player input were actually better off automated. Another big change was the introduction of the SpriteState class which we used to help animate the sprites in the game. Another addition was the use of Strategy in the final game to help with weapon behavior. In general we can see that the depth of the final product was a lot more than first expected in the initial UML. The overall structure of the game did not change too much though which means that we were pretty well prepared going into the first sprint with an idea of what we wanted to do.

Third-Party vs Original

As the game currently stands, we are using code entirely written by ourselves. Most of the code was created solely for this project, but certain pieces such as player movement were taken from a previous project Liam worked on. All other code was written by us for the purpose of this project.

OOAD Process

Key Design Process Elements:

1. An issue that we encountered in the design process was being on the same page for how our final game would look. We were able to work through this by having some brainstorming sessions together and deciding on some common goals.

2. A design element that we experienced was how to tackle the scope of something so large like a game. I feel as a group we were actually able to handle this really well as we started by creating a base that we could then add features one by one to after so that the whole time we could test working code instead of hoping it all came together at the end.

3. Finally we were all able to see how having an object-orientated mindset while creating something like a game changes a lot about how you code as we found ourselves looking at different patterns that we could use and making sure to the best of our ability that each class was following the best practices for OOAD.