Retro game with a twist (Super Mario Bros.): Money Grabber

UML, activity diagram and game design patterns of this game by: game developer Mick Gerritsen

My game is based of the original Super Mario Bros game. Except you will be a money bag. Everything will control the same as the Super Mario bros game, except coins will have a big impact on the player. The twist will be: You are hungry for coins. You want to take as much coins as possible and those coins you eat, because you are a bag. The fuller the bag will get with coins, the heavier you will be (so you can’t jump as high as without the coins you took and will be slower). This will result in a game that feels like Super Mario Bros, but will be a bit puzzly: A mix of platforming and puzzling.

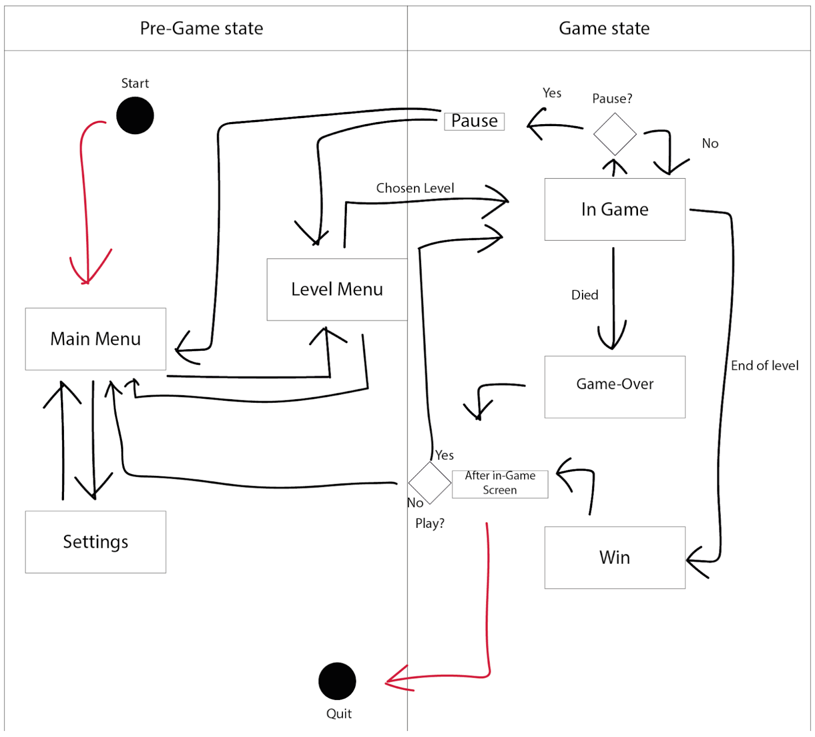
For the game I will use engine Unity 2018 because with this engine I feel the most comfortable of using it. While this engine might not be the best for 2D games, it will be a good option for me because I can easily make prefabs like blocks and enemies and there a few other things that will be handy when making a 2D game. This years version of Unity also comes packed with tools focused on 2D game development like the 2D tilemap tool, cinemachine and 9-slice sprite. The code behind the game will be a composition. I will use raycasts and Unity’s build in 2d physics for the colliders and rigidbodies. To have great control over how the 2D movement of the player will feel, I want to make a raycast playercontroller. With normal Unity physics it can also be done but I don’t want to do that because I want to make the players jump unique and control parameters that I otherwise can’t do unless I mess around with the 2D physics system in Unity that isn’t always as solid. For things as coins and items I will use colliders. The scripts on these gameobject will check for the collision with the player. I will also have a game manager script that will keep track of important things in my scene like the total amount of coins, music and game transition (Game Over or Win screen). I will also have coin weight manager that will convert the current amount of coins in one level to weight (gravity on player controller). This amount of current amount of coins will get passed over to a UI class as well as the all the coins collected in all levels and the weight of the player.

I will use sort of a singleton for my objectpool. That objectpool will hold all the coins that bag eats, and when the bag spits them out, the pool will organize all of that. For game-over or win I want to use events because I am very sure that there will be alot of things that need to work at the same moment. Because those will have seperate methods, I can put them in one event. This is not very high priority for me because this can also be done with scenes in unity.

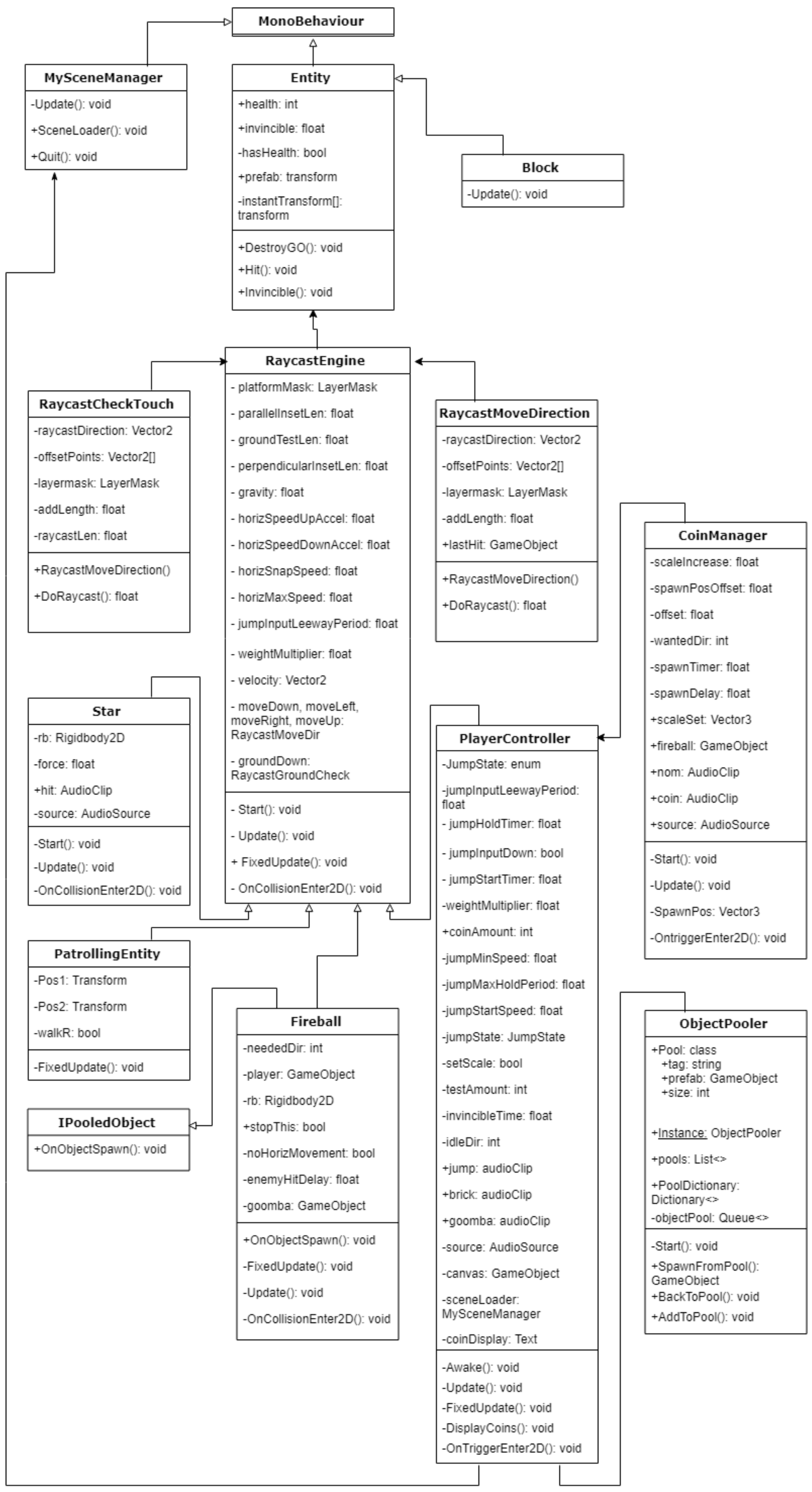
Object pooling is also something that I want to use. For all the coins that will be picked up or spit out it will be useful. Once I have setup the objectpool system, it will be easy to use with one or two functions. The raycast engine I will use for all objects that have a determined movement. Like the spit out coin, because it always needs to have the same bounce and it needs to have a fixed x-movement. For patrolling entities it will also come in handy. This way I will inherit from the raycast engine class and that will feel very organized fast to work once I have set it up. It also reduces the amount of spaghetti code. I will also make an entity class that is abstract. This is the class for a facade game design pattern. This class holds health and a die function. All entities have it but they all behave differently so that’s why.

Priorities (Highest to lowest)

- Raycast-engine
- Coin objectpool
- Enemies
- Animations, audio
- Polish



The way I programmed has been influenced over the weeks. In the first weeks I already knew that I wanted to make a raycast engine for my player controller. But the way I shaped my raycast engine has changed. I thought it was a smart idea to make the raycast engine so I can use it for multiple game objects. I wanted to also make enemies and fireballs so it could be really handy. Eventually I designed my code in a way that I could inherit the raycast engine class 4 times. Otherwise it would cost me a big amount of time, if I made separate raycast engines and for further enemies or project it will be a great building block. Alongside the player controller, I made a coin manager. This class manages everything that happens with the coins in the scene. I wanted to keep this separate so it would be a bit more organized and nice if I changed mechanics up. The object pool is also made in a way with functions to spawn, add or readd game object to the pool. It works with the recognition of tags and will only work with game objects that have included the IPooledObjects interface. This is another case in which I thought about the possibility of adding content in the future and it also worked as a good tool in my game. Every class (except ObjectPooler, CoinManager and MySceneManager) has health so can be destroyed. This is made possible of another layer of inheritance over my raycast engine. This class is called Entity because a lot of objects in my game are entities. Overall my code has become so much more structered and it differs a lot in how I inherit classes. At first I had made separate classes derived out of MonoBehaviour but now it has become a nice base on where I can add enemies, player controllers or even UI and it won't be as clunky as first.


MonoBehaviour
MySceneManager
-Update(): void
+SceneLoader(): void
+Quit(): void
Entity
+health: int
+invincible: float
-hasHealth: bool
+prefab: transform
-instantTransform[]: transform
+DestroyGO(): void
+Hit(): void
+Invincible(): void
Block
-Update(): void
RaycastEngine
- platformMask: LayerMask
- parallelInsetLen: float
- groundTestLen: float
- perpendicularInsetLen: float
- gravity: float
- horizSpeedUpAccel: float
- horizSpeedDownAccel: float
- horizSnapSpeed: float
- horizMaxSpeed: float
- jumpInputLeewayPeriod: float
- weightMultiplier: float
- velocity: Vector2
- moveDown, moveLeft, moveRight, moveUp: RaycastMoveDir
- groundDown: RaycastGroundCheck
- Start(): void
- Update(): void
+ FixedUpdate(): void
- OnCollisionEnter2D(): void
RaycastCheckTouch
-raycastDirection: Vector2
-offsetPoints: Vector2[]
-layermask: LayerMask
-addLength: float
-raycastLen: float
+RaycastMoveDirection()
+DoRaycast(): float
RaycastMoveDirection
-raycastDirection: Vector2
-offsetPoints: Vector2[]
-layermask: LayerMask
-addLength: float
+lastHit: GameObject
+RaycastMoveDirection()
+DoRaycast(): float
CoinManager
-scaleIncrease: float
-spawnPosOffset: float
-offset: float
-wantedDir: int
-spawnTimer: float
-spawnDelay: float
+scaleSet: Vector3
+fireball: GameObject
+nom: AudioClip
+coin: AudioClip
+source: AudioSource
-Start(): void
-Update(): void
-SpawnPos: Vector3
-OntriggerEnter2D(): void
Star
-rb: Rigidbody2D
-force: float
+hit: AudioClip
-source: AudioSource
-Start(): void
-Update(): void
-OnCollisionEnter2D(): void
PlayerController
-JumpState: enum
-jumpInputLeewayPeriod: float
- jumpHoldTimer: float
- jumpInputDown: bool
- jumpStartTimer: float
-weightMultiplier: float
+coinAmount: int
-jumpMinSpeed: float
-jumpMaxHoldPeriod: float
-jumpStartSpeed: float
-jumpState: JumpState
-setScale: bool
-testAmount: int
-invincibleTime: float
-idleDir: int
+jump: audioClip
+brick: audioClip
+goomba: audioClip
-source: AudioSource
-canvas: GameObject
-sceneLoader: MySceneManager
-coinDisplay: Text
-Awake(): void
-Update(): void
-FixedUpdate(): void
-DisplayCoins(): void
-OnTriggerEnter2D(): void
PatrollingEntity
-Pos1: Transform
-Pos2: Transform
-walkR: bool
-FixedUpdate(): void
Fireball
-neededDir: int
-player: GameObject
-rb: Rigidbody2D
+stopThis: bool
-noHorizMovement: bool
-enemyHitDelay: float
-goomba: GameObject
+OnObjectSpawn(): void
-FixedUpdate(): void
-Update(): void
-OnCollisionEnter2D(): void
IPooledObject
+OnObjectSpawn(): void
ObjectPooler
+Pool: class
+tag: string
+prefab: GameObject
+size: int
+Instance: ObjectPooler
+pools: List<>
+PoolDictionary: Dictionary<>
-objectPool: Queue<>
-Start(): void
+SpawnFromPool(): GameObject
+BackToPool(): void
+AddToPool(): void