07.11.2022 – 21.11.2022

SPRINT 2 – Team No. 1

DONE

- Shooting mechanic ⇨ Assigned to Rareș
  - Making bullets
  - Shooting bullets
  - Bullet impact with environment

⇨ Added glow effect using URP and a Volume object with bloom, a Shader Graph that has a material with HDR.
⇨ BulletMovement.cs is a script that moves the bullet and handles interactions:
→ it receives a function from a gun controller that is used to add this bullet instance to a queue when it's deactivated
→ "Action<GameObject> addBullet" is the function "setActionAddBullet()" is the setter used by the controller to pass the function
→ in Start() and also in OnEnable() it starts a coroutine that deactivates the bullet after a certain amount of time
→ in FixedUpdate() it checks if the distance that is going to be traversed by the bullet, with Translate() in Update(), has an object in it's path. For that is used a Raycast() with an offset so that it triggers when the bullet is in the middle of the contact point. If there is something, it activates the startDeactivating procedure
→ in Update(), if startDeactivating is false, then it moves normally in one direction with Translate, otherwise, it stops the movement for performance
→ Deactivate() handles the events that trigger when the bullet hits something, it makes it invisible, it waits for the particle effects to play, then it deactivates the gameObject and adds himself to the queue of the gun controller.
⇨ GunFiringController.cs handles the firing of bullets

→ in Start() it searches for all the effects that need to play when a bullet is fired, and it stores them in one array for easy access

→ addBullet() is the function passed to every bullet to add itself to the queue inactiveBullets

→ in Update() it checks if it can fire a bullet, if so, it plays the effects, and checks the queue for bullets, if it's empty it creates one more, and it passes the function addBullet()

→ waitToFire() and Reload() are coroutines used to handle fire rate and reload time

- Make a gold coin pickable ⇨ Assigned to Amer
  - Update HUD and player gold data

⇨ Added Methods:

→ CoinsController.Start()

→ CoinsController.Update()

→ PlayerController.OnTriggerEnter2D(Collider2D collision)

⇨ The above methods were added in: CoinsController.cs and PlayerController.cs

→ The PlayerHealthController.AddCoins() method was updated to be a static method, in order to be called directly from other scripts/classes without having to create an instance of the PlayerController class.

⇨ Added Coin prefab, along with Coin tag and Player tag(for the player game component).

→ PlayerController.OnTriggerEnter2D(Collider2D collision) gets called when the player collides with an object that is also a trigger. Once called, the method will check if it has collided with a coin and only then will deactivate the coin(to make it disappear from the screen) and call the PlayerHealthController.AddGold(); method to increment the collected gold score. This has the effect of simulating the player picking up coins.

→ CoinsController.Start() sets the initial position of the coin. CoinsController.Update() updates the position of the coin to make it move up and down. The movement is implemented with the help of Mathf.Sin(), which gives the up-down movement effect using the amplitude and frequency attributes of the CoinsController class.

⇨ The overall change is that the player now has coins that he can pick up and increase his gold score.

- Create rooms ⇨ Assigned to Marian and Patricia
  - Make rooms with all possible entries
  - Make different room centers

⇨ Created prefabs in *Prefabs/Rooms/Walls* with all possible variants of door positions
⇨ Created prefabs in *Prefabs/Rooms/Grounds* with 3 different prototype grounds
⇨ Created a demo of a possible scene, combining different walls/grounds
Extra: Added a function *MoveCamera* in *PlayerController.cs* script that handles camera position by the position of player in the scene

- Research how to do the procedural generation for the whole level using all types of rooms ⇨ Assigned to Rareș

⇨ LevelGenerator.cs generates a map using a simple randomized algorithm. It has a probability of succes, let's say 0.23, and if the random number is below 0.23 the map expands with a room, otherwise it doesn't. It decreases the chance the further you expand. Every room has it's own chance of expanding.

→ in Start() the algorithm generates the map, first it creates the rooms using a BFS style of algorithm, then it uses the Build() function to assign the proper prefab. the AddRoom() function adds the new room to the array rooms, and then assigns a key of the same coordinates as the room, the position of the room in the array as it's value. This is done in order to find any room in the rooms array by the coordinates in O(log n) time.

- Make a breakable box ⇨ Assigned to Marian
  - Give the box a chance to drop a gold coin

⇨ Added in *SampleScene* a breakable box for testing

⇨ Created *BreakableBox.c*s script that has 2 methods:

→ *HitBox()* - a public void that gets called from *BulletMover.cs* when the bullet hits the box collider

→ *BreakObject()* - a private void that gets called when *bulletCounter* is equal with *nrOfBulletsToDestroy* variable and it makes the list of box parts to activate their Rigidbodys for visual response and deactivates the Sprite Renderer and Box Collider 2D of the main object

⇨ In *BulletMover.cs* script added *HitBreakableBoxIfHit()* method that checks if the object that hits the bullet is a breakable box and if it actually is then calls the *HitBox()* method from *BreakableBox.cs*. Also, added the *hitBox* bool for not calling the function more than once

NOT STARTED YET

- Search for room and character assets ⇨ Assigned to all

⇨ It's not mandatory for the prototype of the game.