Documentation – 2D Platformer

Before making the game, we gathered ideas on the topic we wanted to have for our Platformer. We did a little brainstorming together and thought about the power up possibilities. We kept the time we got to finish this platformer in the back of our head to not make too big plans which we wouldn't be able to finish or crunch too hard. After deciding on an idea which we both liked, we started to organize our Codecks Board. Again, we had a clear division of the work. Alessa did the coding and the implementation of the animation and sprites into Unity and Annika did the Art and animations for the game. We both did the level design on different levels.

The game theme that we decided on was a dark cave with a young vampire girl as the player character. The little vampire girl is having a nightmare, in which she is trying to dodge the evil garlic and finds comfort in collecting blood drops. We wanted to show the backstory of the vampire girl a bit more but because of time issues we hope to manage this in the next semester. We also have the idea to implement a little shoutout for the players to donate blood and support blood donations. A sneak peak can be seen in the end of the second level with the lifeline of blood drops. Hopefully we can implement this idea and share it also on itch.io to raise attention on this topic.

Other things that we would love to implement are more enemies for the player to fight or dodge and a better fight animation. Also, an animation that turns the character into a bat when pressing spacebar twice and makes the flying effect a bit more visible and gives the vampire character more depth. Like mentioned before, we want to implement more backstory of the character, which we would do with a short cutscene in the beginning. In this cutscene, the grandmother of the vampire girl reads a bedtime story to the her, which is about the evil garlic. This gives the player a real reason in the game to fight the garlic and fear it.

After Annika created the Unity project, we realized shortly after that, even though she picked the 2D options, it didn't seem to be a 2D template. Annika already worked on the project without realizing this at first but gladly Alessa did before working further on the project. We then started a whole new project and a new repository for the task. This Unity project gladly worked out fine and we were able to start working. Annika started off with designing the basic character and bringing it to life with an idle animation. After that she created the basic blocks that were needed to build the platforms and corners. Because we both decided that we want to have enemies in our game, Annika created the enemies shortly after, so Alessa was able to test them out in Unity and work with the enemy movement and interactions. Afterwards, Annika created the other Character movement animations like walking, flying, falling and fighting. Adding some more life into the game, Annika created some decorations like flames and waterfalls. To not make the sprites too static, she created an animation for each object in game. The colour palette was inspired by the basic 'vampire movie colours', dark colours like grey, and purple. In contrast to that the decoration is coloured in a bright green to make them stand out but also let them seem natural or

positive and not as a thread to the player. The garlic enemies and the turret are also both in high contrast to the background, which makes them clearly stand out. Because the friendly ‘ghost helper’, which leads the way to the player, is also coloured in white, Annika gave the sprites a facial expression, to make clear which one is a thread to the player. The background of the main menu of the game also gives a little insight to the player what the game will be about, with the same colour palette, which already sets a dark mood and shows two bats flying in front of the full moon. Creating a bit more dynamic in game, we decided to have a moving background in game which gives the background more depth and creates the feeling to be inside a cave. Annika drew the different layers for the background and Alessa implemented these in Unity. As well for the music, which are the only assets we used, we choose a bit scarier or horror themed music in game, to fit the nightmare idea. For the UI sounds we used retro game sounds which fit the pixel art. Annika searched for the music and sounds in the Asset store and Alessa implemented these in Unity.

With working on this project, we learned even more about Unity itself as a program and how to use it and its functionalities. Annika was excited to try out pixel art for the first time and to do animations as well, which she learned more about and was able to dive deeper into this topic. Making animations was fun to her and she is looking forward to getting even more into it in the future, not depending on the art style. Alessa was able to get even more into coding and learned more about saving and loading, as well as the player interaction with enemies. She also learned more about the ways of referencing other scripts and methods. We both learned how to build a level with the tile map in Unity.

What worked well was for Annika to get into pixel art, even though she was not able to work on her tablet, where she usually draws, and had to draw on the pc by mouse clicking. The animations were also working well after getting into it and understanding the program she used. The level building worked out fine as well as the implementation of the animations. Alessa was also able to manage the comments of the code and the documentation more smoothly and easier.

Something that did not worked out that great, was the saving and loading for the high score. Alessa was talking to other classmates in order to get it done, but that sadly that did not help in the end. With the support of Hanna she explained her scripts to me and I added with small changes in our game, but still we could not manage to get it working. So Alessa decided to delete the high score related objects and scripts out of the scene and comment the scripts out in order to avoid errors and warnings.

Sadly, Alessa also had internet connection issues again, which took away some time from us.

All in all, it was a great and fun experience to create this platformer. We really are looking forward to work further on this game and implement our ideas into it. Which was a bad

frustrating, was the little time we had and all the other university work in between, which didn't let us focus on this project as much as we would have liked.

General:

Coding wise I definitely learned that I should not start working on the code when being in a rush and to try working cleaner from the beginning. It helped me to keep the project more organized, keep a better overview and avoid careless mistakes. Furthermore, I noticed that it is a lot easier to do the coding documentation and commenting of the code while or shortly after finishing it, because I do not have to get back into understanding what I did days or weeks after. The same goes for small errors or mistakes, which I decided to fix in the end but turned out to take a lot more time than it would have when I would have fixed them immediately. I learned a lot about animation and implemented them in our project for the first time. Also, I got to learn more about the ways of referencing other scripts and methods, even though it is still a bit confusing to me sometimes.

Sadly, there is still a lot of stuff missing in this project, especially because I was still struggling with my internet connection in the beginning and lost a lot of time because of that. For example I would change the way of implementing the sound effects, because it seems to be pretty messy and complicated to me, so I would start on working on a sound manager. I am also planning on changing the CameraFollow Script and add clamping and a bit of a delay to it to improve the gameplay experience. Furthermore, I would add better animations and some more mechanics, because we kept it as simple as possible this time in order to get the project and the other projects for university done in time.

Scripts:

Camera

CameraFollow:

I decided on keeping the following of the camera as simple as possible in order to save some time. Therefore, I created a Transform target variable which will be set to the Transform of the player in the Inspector and is the reference to what the camera is going to follow. Also I added a Vector3 offset to set an extra distance to the target, so the camera does not snap directly to the position of the player later on. Then I used LateUpdate instead of Update to set the new position of the camera because I want to make sure that the target is already done with its movement when the camera follows in order to prevent jittery behavior.

Parallax:

First I created two variables, which I need to get to know the length and the start position of the sprite for the Parallax in the Start function by setting startPosition to the x position of the object the script is sitting on and finding the length of x by getting the SpriteRenderer component. Furthermore, I set the variable to create a reference to the camera in the Inspector and a last variable which will define how much parallax effect will be applied to the different sprite parts. In Update I created a distance variable, which is how far the object has moved in the world space. After that I am moving the sprites by setting a new Vector3 and adding the distance variable to the startPosition. In order to let the background repeat itself I created a temp variable, which is defining how far the sprite was moved relative to the camera. If the temp is larger than the position, we started at I am adding the length or subtracting the length from the startPosition again.

Collectables

BloodPickup:

The BloodPickUp script functions as a script for the blood drops the player has to collect during the game. The only variable I created is a ScoreCounter variable, which I need in order to create a reference to the ScoreCounter element from the UI and its ScoreCounter Script. Furthermore, I created an AudioSource and an AudioClip variable, which will be assigned in the Inspector.
In OnTriggerEnter2D I check if the object colliding with the trigger of the blood drop is the player. If that is true I play the audioClip assigned once and increase the score of the ScoreCounter with the ScoreCounter script by one. After that I destroy the object so the player is not able to collect the blood drop again and increase its score several times.

PowerUps:

Here I also wanted a sound to get played when picking the Item up. The same goes for attacking, taking damage, the opening of the PauseMenu etc. Since I used the same way to play sound in every script I will leave the sound parts out.

PowerUpBlood:

The Blood PowerUp is doubling the score of the blood drops the player has collected.
I created an OnTriggerEnter2D method and checked if the player is the object colliding with trigger. In that case I am calling the PickUp method following afterwards. In the PickUp method I am destroying the game object the script is sitting on, so the player is not able to collect it again and reference the ScoreCounter in order to being able to change the value of the score by two or doubling it.

PowerUpHealth:

The Health PowerUp regenerates some of the player's health in case he is damaged, otherwise it will not have any influence on the player. To increase the player's health by a specific amount I created a new int variable which can be set in the Inspector. After that I created a healthBar variable to reference the healthbar and call the PickUp function in the OnTriggerEnter2D method like in the other PowerUps Scripts. In the PickUp Script I set a reference to the Player Script on

the player game object in order to access the current health of the player. After that I added the extra health to the currentHealth of the player in order to increase it. Finally, I needed to access the SetHealth method of the HealthBar again, because I noticed that the value or the color of it does not get updated like the current health of the player in the Inspector.

PowerUpSpeed:

Since the player is collecting the blood drops and is also rated by time, we decided to let the player walk faster for a specific amount of time. ExtraSpeed is the value the movementSpeed of the player will get increased by. In OnTriggerEnter2D I needed to call the Coroutine because I created an IEnumerator instead of a normal function, because this PowerUp is just a temporary improvement. So in the IEnumerator PickUp I referenced the movement script of the player and added the extraSpeed on top of the moveSpeed. After that I only disabled the SpriteRenderer and Boxcollider2D because if I would destroy the Object immediately the whole function would stop. After that I wait or let the function run through for 10 seconds, decrease the moveSpeed by the extraHealth again and destroy the PowerUp in the end.

Enemies

Spikes:

The only variable I used is a player variable to reference the player in the Start method and make the player health and its functions accessible to the Spikes Script. The OnTriggerEnter2D method gets called when a collider enters the trigger. If the collider that enters the trigger has a “Player”-tag, I am calling the TakeDamage function from the Player Script and set the value of the damage the player has to take in order to reduce the player’s health. Also I am adding the Knockback IEnumerator by starting a Coroutine and set the parameters defined in the Player Script. As a knockbackDirection I am just giving the players position and the knockbackDuration is set to a very small number so it is just a very short time for the player being knocked back.

Turret:

The turret was probably the most complicated enemy to create. Especially because the attacking does not work the way I wanted it to work with the turret. I think it is this way because of the colliders, which I confused it with. So in the end I had to enable the isTrigger box on the turret in order to let the player damage it. But this is fine, because the turret hides behind the ground or is behind the player anyway.

TurretAI:

First I am setting two int health variables for the turret so the player is able to attack the turret and reduce the health later on. After that I used several different variables, sorted by their type, which are mainly used for getting to know if and how the turret should start shooting and for setting references to all the different components needed for the functions.

In Awake I am creating the reference to the animator in order to be able to call the animations in the functions following afterwards.
In Start I am setting the currentHealth of the turret to the maxHealth, so the turrets health is completely full at the beginning.

In Update I am setting the parameter of the Animator and call the RangeCheck method and destroy the turret when the health is equal or smaller than zero.
The RangeCheck function I first set the distance float to the distance between the turret itself as point a and the target as point b. If the distance is smaller than the wakeRange set in the Inspector, the turret will be set to awake and the other way around. So I used this function to get to know if the player as the target is actually close enough to let the turret wake up.
After the RangeCheck I created a public Attack function, which is used to define when and how the turret should start shooting. It gets called from the AttackCone Script and takes a Boolean AttackingRight as parameter, which will be important to set the right cone the turret is shooting from. First I let the bulletTimer increase every second by adding it with Time.deltaTime. In the following if statement I am checking if enough time has passed for the turret to shoot another bulletClone by saying that the bulletTimer, which is increasing over time, has to be greater than the shootIntervall set. If that is the case I am setting a new Vector2 direction, which will return the direction towards the target and normalize the Vector after that. Then I take a look if the player attackingRight is true or false, later set in the AttackCone Script and create a new bulletClone dependent on the position of the player and cast it as a game object. After that I am getting the velocity of this bulletClone, so I can assign the direction and the speed with which it is going to fly to it. Finally I set the bulletTimer back to zero, so it can start counting and shooting again.
At the end I created a TakeDamage function again, which can be called from other classes and is used to reduce the health of the turret by dmg, set in the classes where it gets called.

AttackCone:

The AttackCone Script is there to let the turret actually start shooting if the player is in the AttackRangeTrigger. The TurretAI variable is helping to reference the turretAI script on the turret game object in the Start function, which is the parent of the AttackRangeTriggers on which this script is sitting. The isLeft Boolean is used to check if it is the left or the right AttackRangeTrigger the turret will be shooting from.
Then I put an OnTriggerStay2D function in the code, because I want the turret to keep on shooting as long as the player is in the trigger. Then I am checking if the object being in the trigger is tagged with “Player” and if that’s the case I am calling the Attack method from turretAI script and set the Boolean to attackingRight to false or true depending on which Cone it is and to call the matching part of the if-statement in turretAI to let the bullet shoot from the right point.

Bullet:

I created an extra Bullet Script so the bullet which gets shot by the turret actually has an influence on the player and let it disappear so it does not stack up together with the other bullets. When the object with the Bullet Script on, collides with an object that is not marked as trigger and has a “Player”-tag on, the TakeDamage function from the Player script gets called and reduces the health of the player by the amount set. I needed the collision set to not to be trigger or have a “Turret”-tag because otherwise the bullet would collide with the AttackRangeTrigger or Turret collider before it reaches the player and would get destroyed first. After that the bullet gets destroyed and will be removed out of the scene.

Patrol:

The Patrol enemy is one of the enemies which is vulnerable and therefore I created the basic two health variables I also used for the other objects, which are able to take damage. Then I added a moveSpeed variable which can be set in the Inspector and defines how fast the Patrol is going to move. I also added two float variables which set the length of the ray which will be created. The Boolean movingRight is used to tell the patrol where he has to move to after reaching the end of a platform or a wall. Furthermore there is a Transform groundDetection, which references the groundDetection object from the Patrol by setting it in the Inspector.
In Update I simply used transform.Translate without any Rigidbody to move him to the right based on the speed and Time. In order to let him also move to the left whenever he reaches the end I created a ray called groundInfo and a ray called wallInfo, so he is not only detecting the end of a platform but also of a wall. As parameters I set the origin from the groundDetection, the direction to which it is shooting out a ray and how long it should be. With the if-statement I am checking if the ray hits a wall or does not hit the ground anymore. If that is the case I turn the moving direction about 180 degrees so the Patrol faces the other direction now and therefore also change the movingRight Boolean. First I only had a groundInfo ray but since I wanted to also detect a wall and turn around I added a second ray which is not directed downwards but to the right. Also I added the same if-statement as in the turretAI Script in order to let the patrol die. Then I added the same function like in the Spike Script to let the player take damage, bounce back and blink red when colliding with the Patrol and the method to let the patrol take damage himself.

Menus

The Menu Scripts are almost the same Scripts used in the 3D Walking Simulator Project. The only difference is in the MainMenu Script. Here I deleted the Play method to start the Game, because I decided to add a scene transition with a crossfade. The crossfade has an own script with a function to call a next scene, so I used this method to call from the start button.

Player

Player

The first variable I set was the currentHealth variable, which is public because I need to access it from other scripts in order to reduce and update the health of the player. After that I put a maxHealth variable, which is mainly used to set the health back to maximum every time a level gets loaded and to make sure that the player does not generate more health than possible when collecting a PowerUp. The last variable I set is to create the reference to the healthBar in the Game and set to maximum and refresh it in Start.
After the first if-statement in Update I set a second if-statement, which is basically a die function, but instead of destroying the player the active scene in the Build Index gets reloaded in order to let the player restart the level.

In order to let the player take damage and to make it visible to the player I created two public functions, which can be called from other classes. The first function is the TakeDamage method,

which is used to reduce the health of the player. It takes an int dmg parameter, which can be set in the other classes individually in order to make a different power of the enemies possible. Then I am reducing the currentHealth of the player by that value set in the other scripts. After that I am playing the red flash animation by referencing the Animation component from the player. Normally I would try to set the animation up like the other ones for the player by setting different parameters, but I decided that this works fine for just one animation and saves me from wasting time figuring out a different way.

The second one is an IEnumerator called Knockback, which is used to push the player back a little bit when colliding with an enemy. I decided to use an IEnumerator in that case, because I wanted to call it on the frame when OnCollision starts, let it run through and let it end afterwards. The knockbackDuration is used to define how long a force is added, the knockbackPower to set how strong the knockback is going to be and the knockbackDirection to set the direction the player is getting pushed to. The timer variable is set to zero and is used to count the time which has passed since the function started. The while loop after that is checking if the knockbackDuration is greater than the timer and while that is the case it is increasing the timer, and is adding a force to the Rigidbody component on the game object, based on the knockbackDirection and power. After that I am ending the function with yield.

I was struggling with getting the knockback part done quite a bit and know that is probably a pretty complex way to do it. But since I got very confused with the Vectors, adding them and referencing them from other scripts I decided to simply use a solution found on the internet and modifying it just a little bit, even if it is not totally functional.

PlayerMovement:

The PlayerMovement Script is used to make the player move around in the game and to set the Animation States. The moveSpeed variable and the jumpForce variable define how fast and how high the player is able to walk or jump. I made the moveSpeed variable public, because I need to reference it in my PowerUp Script in order to modify it. I also added a Rigidbbody2D variable which is set to the Rigidbody on the Player object and the same for the Animator.
In Update I call the Jump function and created a Vector3 movement variable, which is set to the Horizontal Axis Input, predefined by Unity. Y is set to 0, since the movement is only meant to be the left and right movement. After that I use the movement variable and add it multiplied with the moveSpeed and Time.delta Time to the position of the player. I multiplied it with Time.deltaTime in order to make the movement independent from the framerate. Then I check the value of the Horizontal Input, because I wanted the player to face to the direction it is moving. So if the Input is below -0.1f x of the Scale of the player is switched to -1. I decided not to use the Rigidbody for the left and right movement, because the player kept on sliding for a bit and this seemed like an easy solution for us. The last thing I did in Update is to call the SetAnimationState function to keep the animations matching to the current moves.
The Jump method checks for the pressing of the Jump button, which is set to the Space Bar. If the SpaceBar is pressed the jumpForce set in the Inspector gets added instantly. Since the player is playing a vampire we decided not to limit the amount of jumping in order to create an impression of flying. The SetAnimation function sets all of the parameters for the animator in order to play the matching animations. The first parameter I set is the Speed parameter. I set Mathf.Abs in

front of the Input so the animation also gets changed correctly when the player is walking to the left or is giving a negative Input. After that I check for the vertical velocity of the Rigidbody. If the velocity is equal to null the flying and falling animation is set to false, so the player's state can be set to walking or idle. If it is greater than null, the flying animation is active and if the velocity is smaller than null the falling animation will be active.

PlayerAttack:

The PlayerAttack Script is sitting on the Player object and is helping to define when the player can attack and prepares everything to call the TakeDamage function from the AtackTrigger Script. The first variable created labels attacking as active or inactive, which is also necessary for the attacking animation. Both attackTimer and attackCoolDown are needed to create some sort of cooldown later on, so there is some time between the attacks. After that I set the variables for the references to the attackTrigger under the Player object and the animator on the Player object.
In Awake I am creating the reference to animator and deactivate the attackTrigger, so the Player is not automatically attacking when colliding with an "Enemy"-tagged object.
The first if-statement in Update is checking if the player is pressing the left mouse button and if the player currently is not attacking. If that is true, attacking will be set to true, because this is the call to start attacking. The attackTimer is equivalent to the attackCoolDown and the attackTrigger gets activated. Then I check if attacking was activated through the method before and create a cooldown by decreasing the attackTimer over time when it is set to greater than null. Otherwise when it is already null or smaller than null, attacking will be set to false and the attackTrigger gets disabled, so the player is able to attack again. At the end of Update I call the Attacking animation so it is visible to the player that he is doing something.

AttackTrigger:

In AttackTrigger placed a dmg variable, which is setting how strong the attack of the player will be. After that I call an OnTriggerEnter2D method, which will call a TakeDamage method if the object colliding with the Trigger has an "Enemy"-tag. Since there are different enemies and I cannot reference every single one of them to call the Takedamage method on them, I used SendMessageUpwards.

SceneSwitch

LevelLoader:

The LevelLoader Script is sitting on the Crossfade game object and is used to have a smoother scene transition. First I created a reference to the Animator of the Crossfade transition. I created a LoadNextLevel function in which I put the IEnumerator, so I can also call the IEnumerator from the play button in the Main Menu. The IEnumerator LoadLevel starts the animation first, wait for one second and loads the scene set in another function.

SceneSwitch:

SceneSwitch is used by the gates at the end of a level in order to get spawned in the next one. After setting a reference to the LevelLoader Script on the Crossfade object I was able to call the IEnumerator from the other script and set in the buildIndex + 1 to load the next level.

Sound

ButtonSound:

The ButtonSound Script is also the same as in the Walking Simulator, so I will leave it out of this documentation since no code was changed in it.

UI

HealthBar:

First I created a Slider variable to set the value of the slider in the SetHealth and SetMaxHealth function. I also needed to add a Gradient in order to being able to change and set the color of the fill of the HealthBar, to which I create a reference afterwards.
In SetMaxHealth I set the Sliders maxValue to the health given at the beginning of the players Script. Also I set the color gradient to one, since I know that it has to be completely filled.
In SetHealth I am setting the value of the slider to the health again and let the gradient evaluate the color of the slider.

ScoreCounter:

The ScoreCounter as well as the score variable gets referenced in the BloodPickUp Script. In this Script I am only creating the reference to the scoreCounter text object to make to score visible to the player.

Timer:

The first variable is for the startTime, which is set in the Start function to the time which has passed since the application started. The second variable helps to create the reference to the TimerText object in the canvas.
In Update I am declaring a float t which is setting the time since the timer has started. Because I need strings in order to display the time in a text field I created to strings called minutes and seconds. To convert the seconds into minutes I devided them by 60. I removed the decimal numbers by putting an int before the time and transformed them into a string by simply using

ToString. To remove the decimals from the seconds I used “f0” as a parameter. Because I did not like that the seconds and minutes from one to nine are shown without a null in front, I created an if-statement which sets an extra null in front of the time in case the length is equal to one. The last line of code finally sets the timerText to the minutes and seconds declared before.

Assets used:

In game music:

https://assetstore.unity.com/packages/audio/sound-fx/horror-elements-112021#content

UI Sounds:

https://assetstore.unity.com/packages/audio/sound-fx/retro-ui-sounds-48837

https://assetstore.unity.com/packages/audio/sound-fx/minimal-ui-sounds-78266#content

Hit Sound:

https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116

Powerups:

https://assetstore.unity.com/packages/audio/sound-fx/achievement-sfx-free-91639