Abschlussarbeit

Kevin Gregory Agwaze

github.com/RebornRider/S4G-Abschlussarbeit

1 Introduction

1.1 Vision

Create a third-person action RPG Paladin character that can move and attack an enemy in a test arena.

1.2 Goal

I will research already existing Unity character controller assets and extrapolate best practices from their design. Then I will prototype in order a test environment, movement, and controls, camera, animation flow, melee attack, an enemy, UI, sound.

I will not create meshes, animations and sounds myself but resort to external assets.

The inspiration for look and feel is Geralt of Rivia from The Witcher 3: Wild Hunt.

1.3 Technologies

Unity 2017.1.p4

git version 2.9.0.windows.1

Github

GitKraken Version 3.0.0 64-bit

Microsoft Visual Studio Community 2015 Update 3

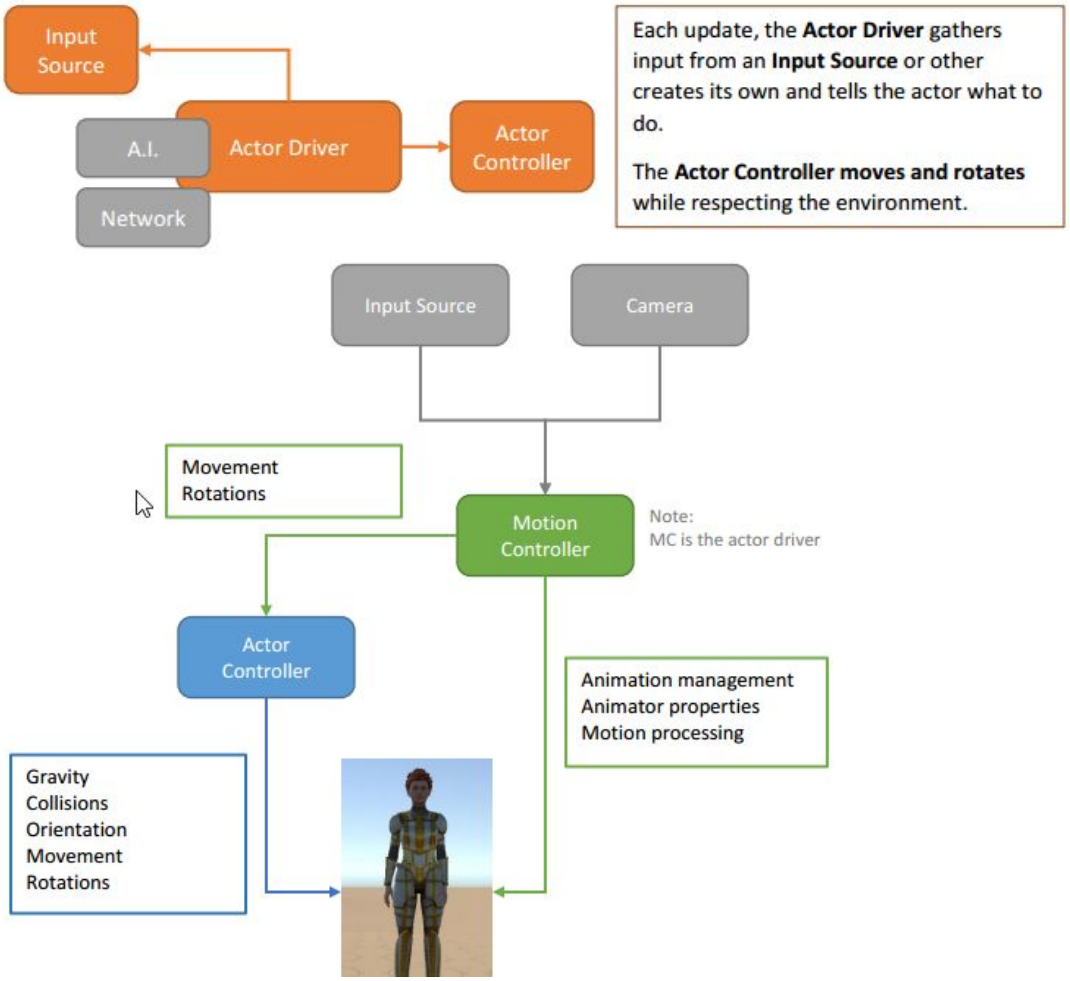
2 Main

2.1 Reference Assets

2.1.1 Standard Assets - Third Person Character (SA-TPC)

The Standard Assets Third Person Character is a simple, input based system made out of two components. The *ThirdPersonCharacter* responsible for moving the character as well as managing Animations and the *ThirdPersonUserControl* responsible for deciding when and how to move the character. It also relies on the *CrossPlatformInputManager* to abstract input from various input devices and platforms.

2.1.2 Third Person Motion Controller (TPMC)



This asset is an animation framework and character controller for any character and any game. It is an “input based” controller that puts user input and responsiveness above physics.

The main components are:
The **Input Source** which gathers input from keyboard, mouse, gamepads, etc. This can wrap Unity’s input system or a third party input system.

The abstract **Actor Driver** determines how the *Actor Controller* moves based on user input, AI, etc.

The **Motion Controller** is a concrete *Actor Driver* and also manages animations, sound effects, *Motions*.

The **Actor Controlle**r is the actual *“character controller*” that handles movement rotations, collisions, gravity, walking on walls, etc. Basically, the *Actor Controller* knows “*how*” to do these things, but he doesn’t know when to do them or why.

The **Motions** are about animation flow. They don’t replace the mecanim Animator, but work with it to control when animations start, when they transition, and when they end.

The **Actor State** is held by the *Actor Controller* and contains state information for the current and previous frames. In this way, we can do comparisons against position, grounded-ness, velocity, etc.

2.1.3 Third Person Controller (TPC)

This asset is a third person framework featuring a character and camera controller, combat system, inventory management, etc. It is structured similarly to the Third Person Motion Controller.It also has the concepts of *InputSource* and *Motions* (here called *Abilities*). Like *Motions* these *Abilities* contain their own Input and Transition information so like the TPMC there are no explicit Animator transition / animation transitions are handled by script. Unlike the Third Person Motion Controller this character controller does not differentiate between *ActorDriver* and *ActorController*. Bot are combined into the omnipotent, 1500 LOC long *RigidbodyCharacterController* class, that also does inventory, spawning and much more.

2.1.4 Conclusions

Best Practices:

1. Both TPC and TPMC abstract away Unity's input system to allow for input device and platform dependent user input, which also makes it easier to mock user input.

2. All three character assets separate the responsibility of moving the character from deciding when and where to move.

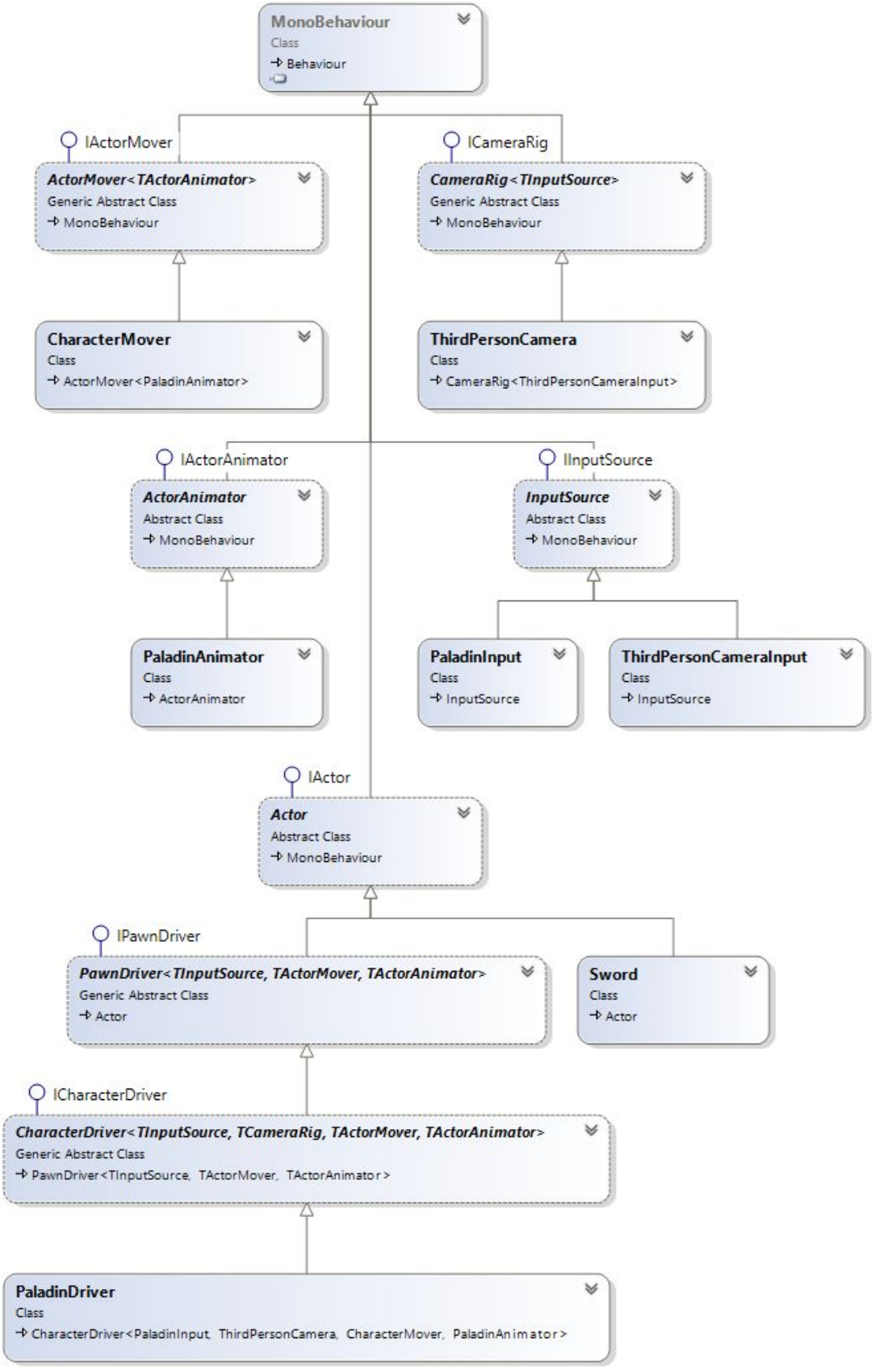
3. Both TPC and TPMC seem to be avoiding mechanim and explicit animator transitions in favor of own solutions.

4. Both TPC and TPMC factor the camera into their architecture.

Creating a new animation flow solution would transcend the scope of this project, but the other best practices seem worth adhering to.

2.2 Architecture / Design

2.2.1 Overview



The Paladin character framework is based on a component based architecture, where each component is based on an abstract base class, which implements an empty Interface to facilitate generic type constraints.

The abstract components of the Paladin character framework are as follows:
An **ActorMover** handles movement, rotation, collisions, grounding, jumping and informs an *ActorAnimator*.

An **ActorAnimator** handles animations and works witch mecanim and StateMachineBehaviours to control animation stare and flow.

An **InputSource** provides two axis inputs and concrete action queries (e.g. WasJumpPressed).

A **CameraRig** is responsible for a camera's position and rotation relative to an anchor based on input from an *InputSource*.

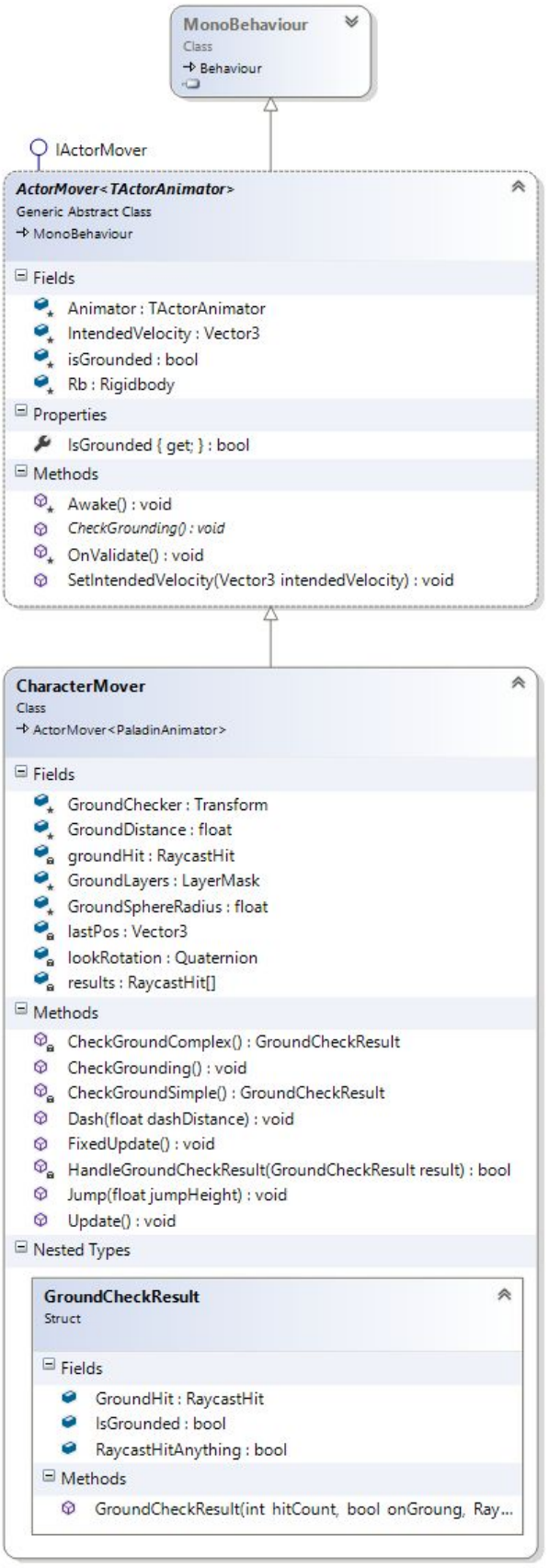
An **Actor** represents a simple in-world entity. It does not need to have a visual representation in the world and does not necessitate user input.

A **PawnDriver** is an *Actor* that controls the animation and movement of a visual entity and has access to user input.

A **CharacterDriver** is a *PawnDriver* that represents a player controlled character with it's own camera.

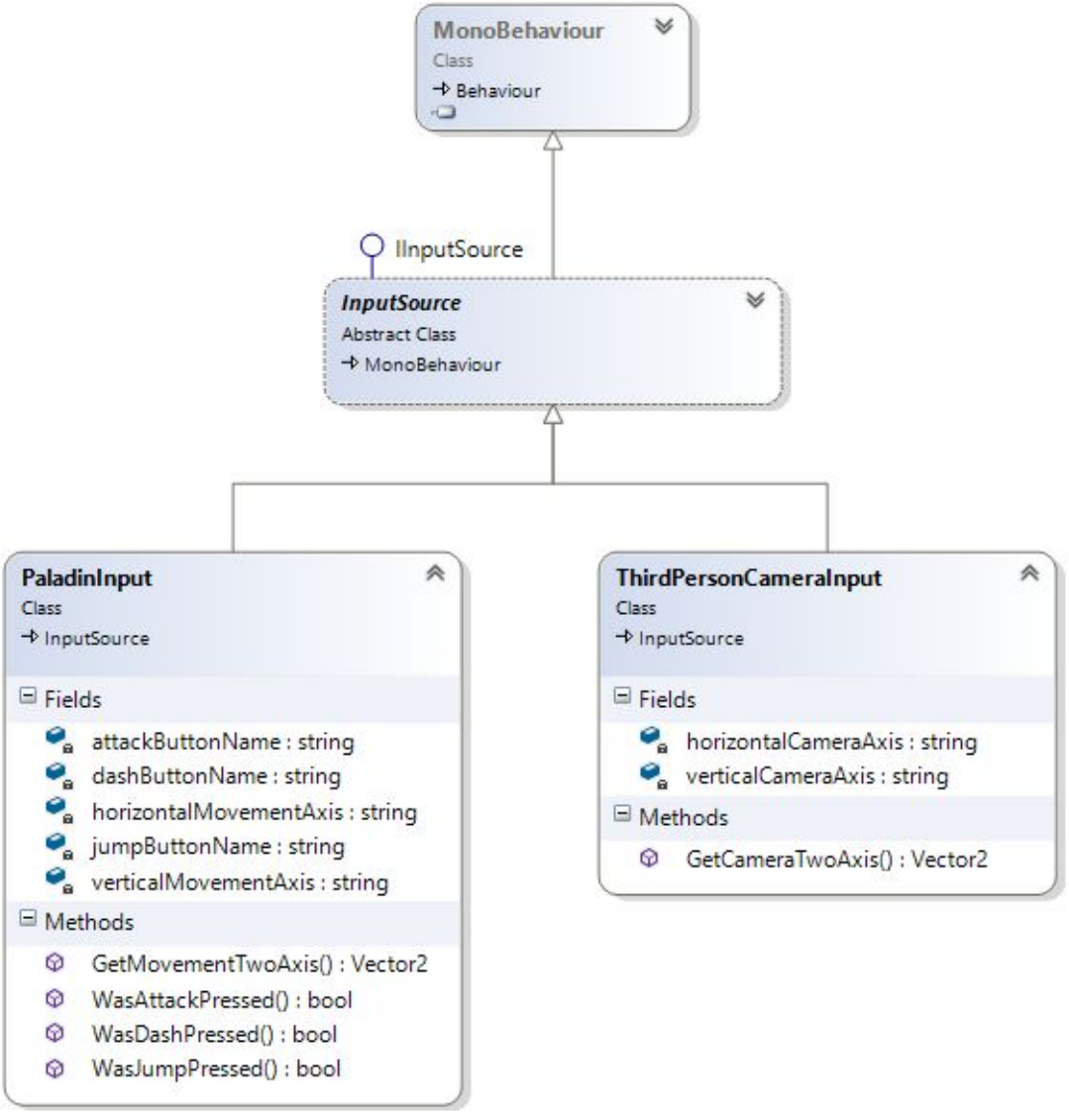
2.2.2 Movement

Movement of the Paladin is handled by the ActorMover implementation *CharacterMover : ActorMover<PaladinAnimator>*. This *Actormover* receives the relative intended velocity every update from the *PaladinDriver* and converts that into a world space movement velocity. The Dash and Jump methods can be called at any time from the *PaladinDriver* to make the rigidbody launch into the air or quickly move forward. Every FixedUpdate the total position displacement is calculated and passed to the *PaladinAnimator*. Then a grounding check is performed, first a simple RayCast downwards and if that does not hit anything a more complex SphereCast is performed. If the Paladin is not grounded, but close to the ground, he is teleported onto it. If the character is grounded at this point we project the relative movement direction onto the surface he is standing upon. If the relative movement on the global xz plane is not zero after that, we rotate the character into towards the movement direction. The Paladin game object needs to have a non-kinematic rigidbody, a capsule collider and transform that indicates from where grounding raycasts are performed attached.



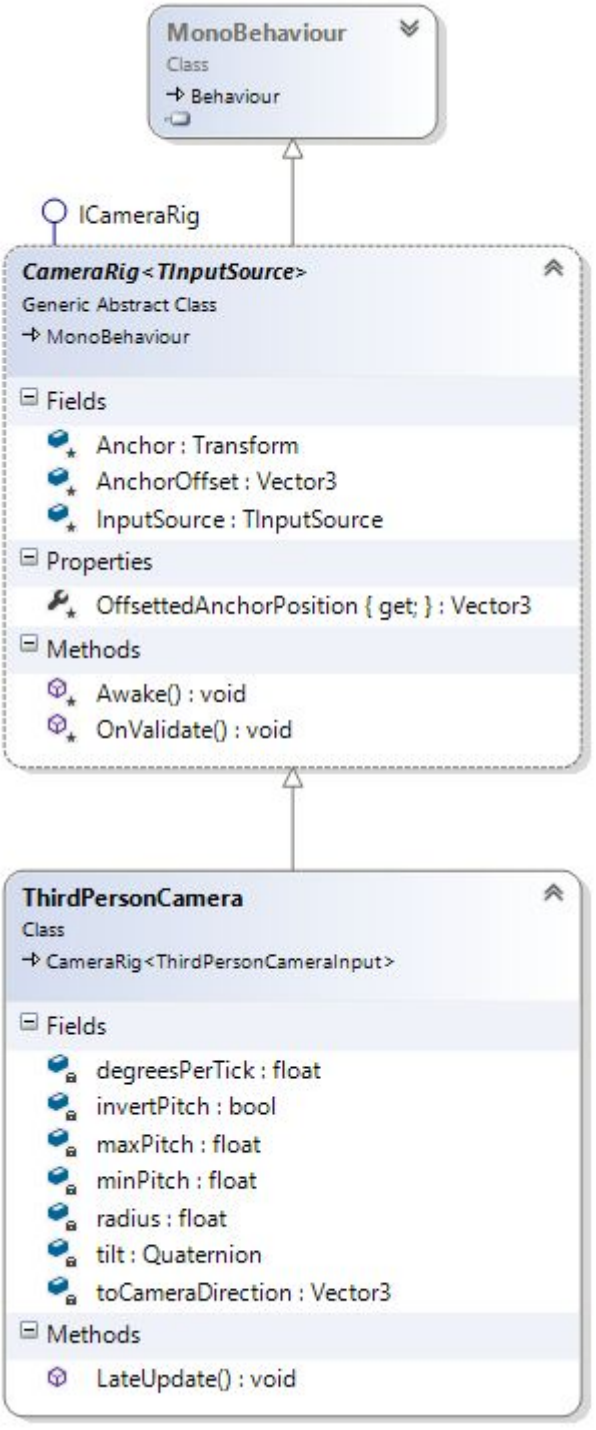
2.2.3 Input

The *ThirdPersonCamera* and the *PaladinDriver* receive user input from the InputSource implementation *PaladinInput : InputSource* and *ThirdPersonCameraInput : InputSource*.
This split of input sources allows input receiver to request exactly and only what they need, following the Interface Segregation Principle.
These *InputSources* Unity's internal input system, therefore the buttons and axes just the names of are inputs set up and customized with Unity's input settings. The default values refer to Unity's default actions and therefore works with every project that has not removed the standard bindings.



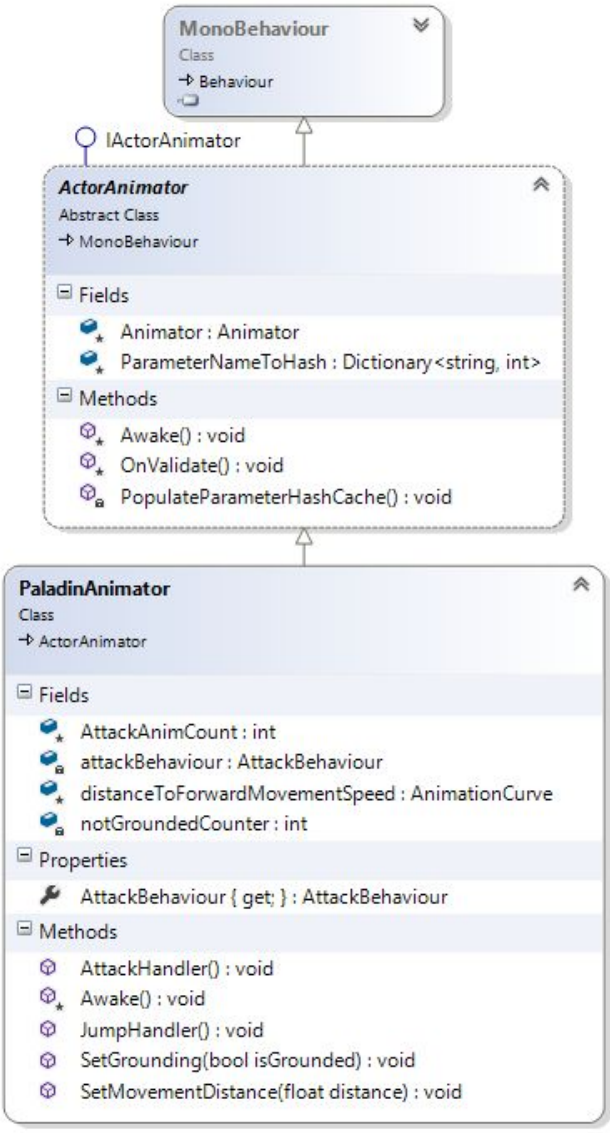
2.2.4 Camera

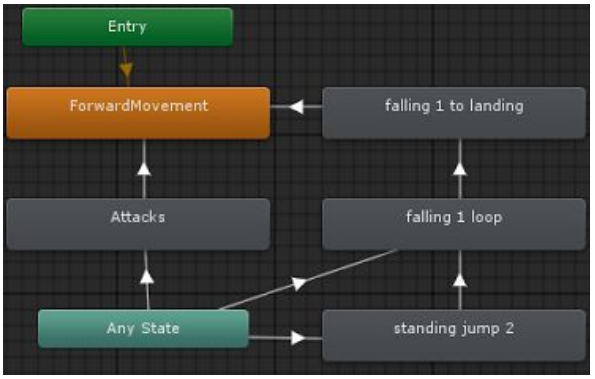
The third-person Camera of the Paladin is handled by the *CameraRig* implementation *ThirdPersonCamera : CameraRig<ThirdPersonCameraInput>*. The *ThirdPersonCamera* orbits around anchor transform's offsetted position during LateUpdate to make sure the Anchor has already moved in that frame. The pitch and yaw of the camera are transformed with user input from the *ThirdPersonCameraInput* (the pitch can be inverted with invertPitch flag). The yaw in unbound, but the pitch should not go over the poles and therefore is restricted by maxPitch and minPitch. Most of the calculations are done in "world up" space and converted to "anchor up" space at the end, to simplify calculations. As movement is relative to the camera, the *ThirdPersonCamera's* view direction is used by the *PaladinDriver* to transform the user's movement input. The yaw turn speed, represented by degrees per tick, is framerate dependent.



2.2.5 Animations

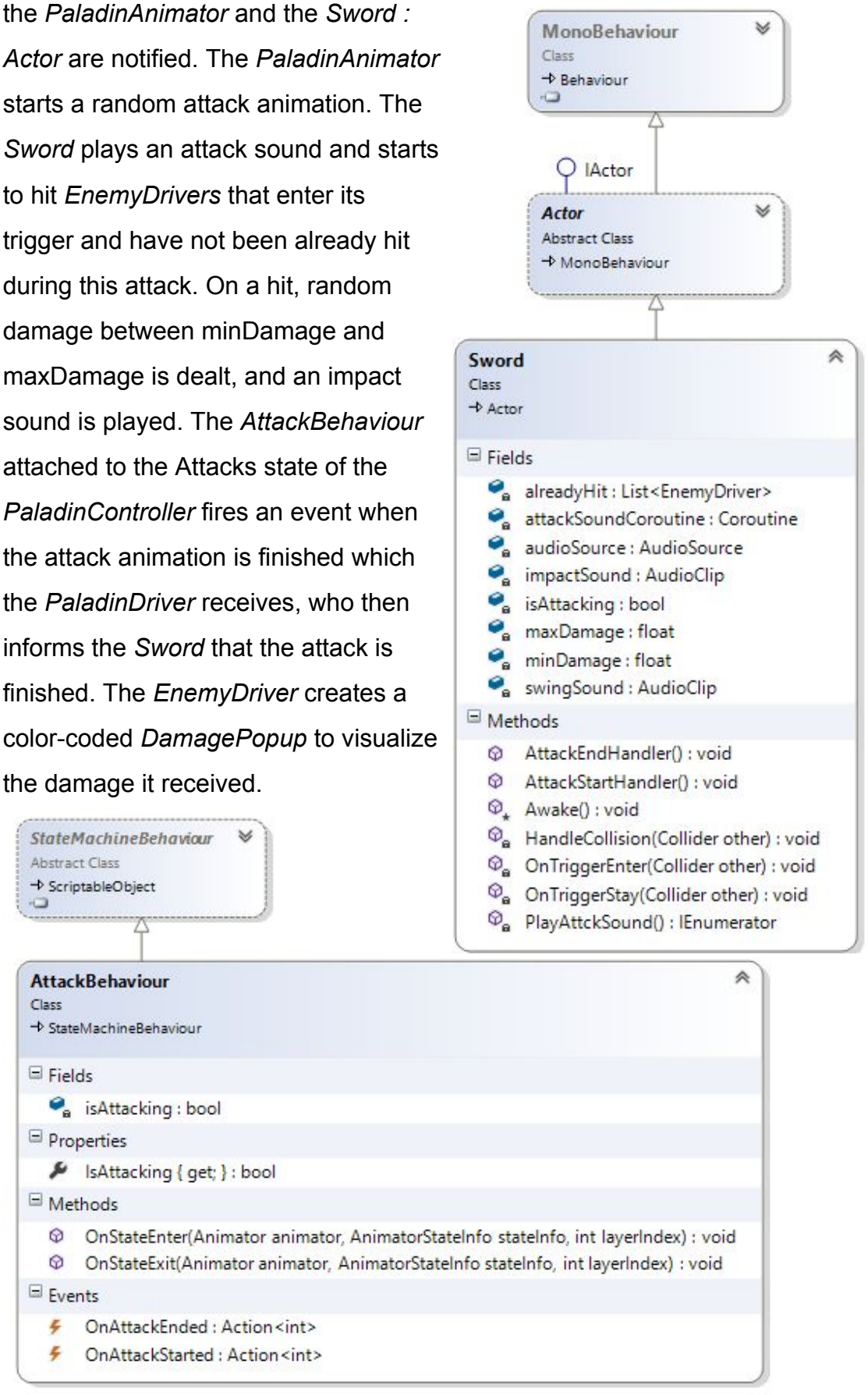
Updating and interacting with a mecanim Animator is handled by the *PaladinAnimator : ActorAnimator*. Most of the animation flow logic is contained in the *PaladinController* which means that the *PaladinAnimator* is mostly responsible for updating AnimatorController parameters. The ForwardMovement state is a Blendtree to dynamically blend between idle, walking, and running animations based on Forward -MovementSpeed.The Attacks state is a Blendtree to play one of two attack animations based on AttackIndex. OnJump triggers a short wind-up animation followed by a falling loop until the character is grounded again. When the character falls without jumping the wind-up animation is skipped. The Attack state has an *AttackBehaviour* attached that informs the *PaladinDriver* over the attack animation progress.





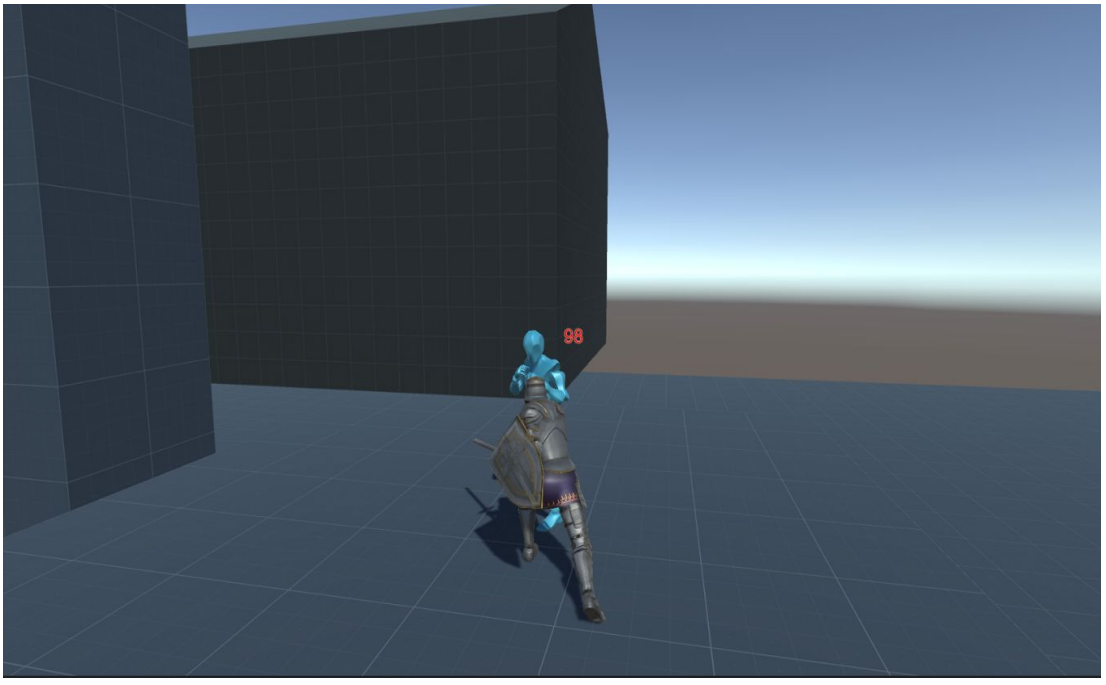
2.2.6 Melee Attack

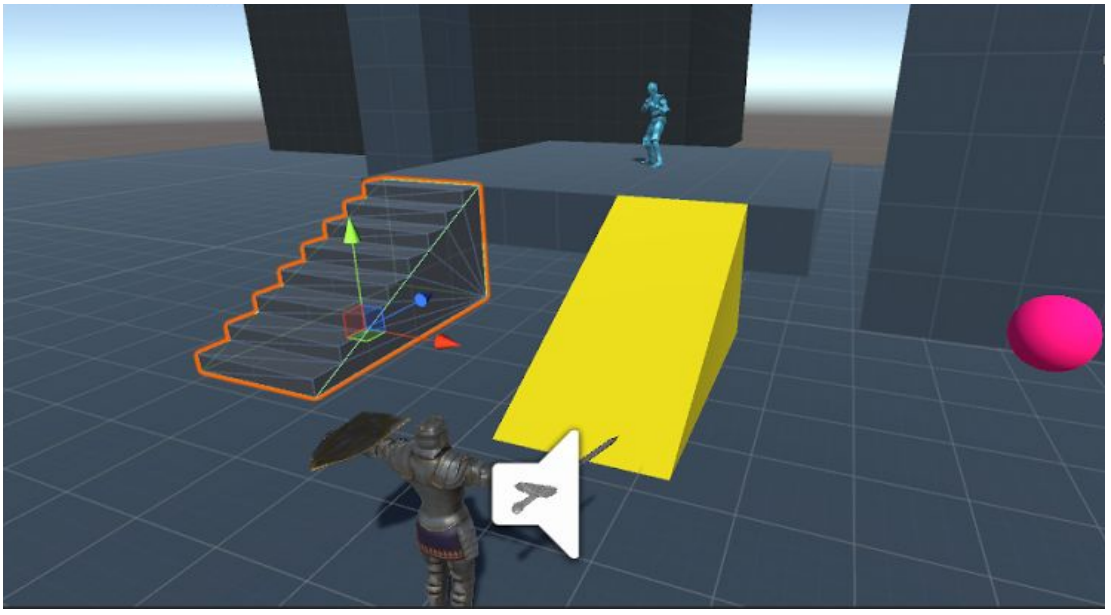
When the *PaladinDriver* receives an attack user input from the *PaladinInput* the *PaladinAnimator* and the *Sword : Actor* are notified. The *PaladinAnimator* starts a random attack animation. The *Sword* plays an attack sound and starts to hit *EnemyDrivers* that enter its trigger and have not been already hit during this attack. On a hit, random damage between minDamage and maxDamage is dealt, and an impact sound is played. The *AttackBehaviour* attached to the Attacks state of the *PaladinController* fires an event when the attack animation is finished which the *PaladinDriver* receives, who then informs the *Sword* that the attack is finished. The *EnemyDriver* creates a color-coded *DamagePopup* to visualize the damage it received.



2.2.7 Test Arena

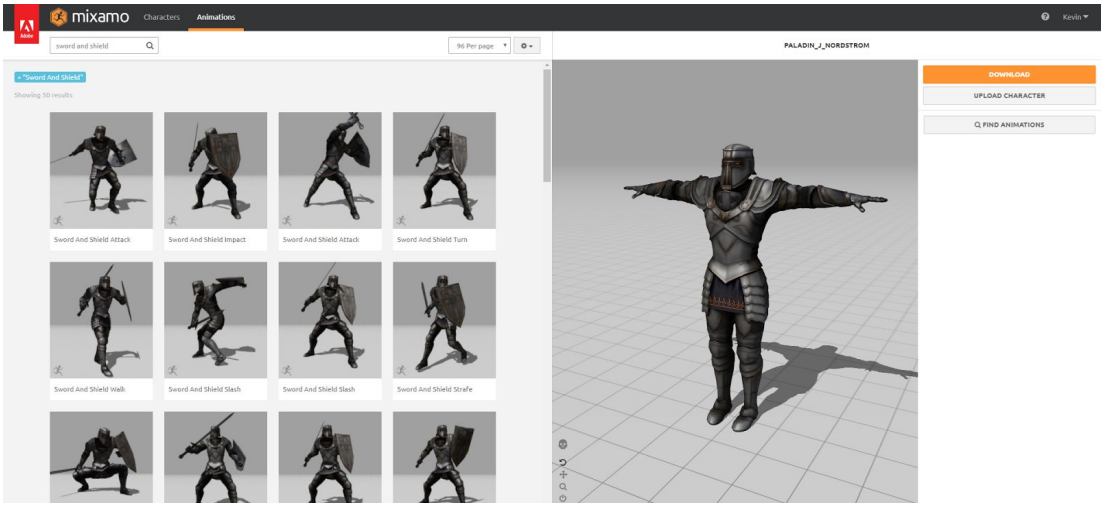
The Test_Arena Scene contains the Paladin character, the third person camera, a basic enemy that can be hit and a traversable environment with a ramp and some stairs (with the collider of the ramp), which allows the testing of movement (walking, dashing, jumping), falling from a platform and attacking. The environment is built from Unity's prototyping package of the standard assets and was used from the earliest tests to completion.





2.2.8 External Assets

The Paladin and enemy mesh plus their animations, are from mixamo.com.



The sounds of attacking and hitting are from Universal Sound FX 1.4

and Kenney Game Assets (Version 1.4)

The environment meshes are from Unity's standard assets.

3 Conclusion

Geralt of Rivia, the main character of The Witcher 3: Wild Hunt, was the inspiration for this character controller. Handcrafted over dozens of man months it is a staple of seamless animation flow, and even they patched in a separate alternative movement mode because "Some players have been reporting that Geralt's responsiveness was not up to their preferences”[1]. Moreover, even after that improvement, a player expressed that ”[Movement] remains my biggest problems with the game. Geralt always seems to take an extra step when you want him to stop. He swings wildly to and fro when you want to turn about. And often it's tricky to focus in on loot or fast-travel signs, or leap up onto your horse.”[2] And just like Geralt the Paladin also has some situations in which he is wonky or an animation looks off. Making the character controller “feel” good I very hard. Myriads of playtesting, user experience design and expertise are necessary to reach such heights. Nevertheless I have create a character that moves, attacks and mostly works in just two weeks. That is something to be proud of.

4 Sources

1. Chalk, Andy: „The Witcher 3 patch 1.07 change list revealed [Update]“ http://www.pcgamer.com/the-witcher-3-is-getting-a-very-large-update-in-version-107/ (2017-09-10)
2. Kain, Erik: „'The Witcher 3' Patch 1.07 PC Review“ https://www.forbes.com/sites/erikkain/2015/07/21/the-witcher-3-patch-1-07-pc-review/#2a273d7c3def (2017-09-10)

- http://www.ootii.com/Unity/MotionController/MC2Guide.pdf
- http://www.ootii.com/Unity/ActorController/ACGuide.pdf
- http://www.ootii.com/unity/motioncontroller/MCMotionBuilder.pdf
- http://www.opsive.com/assets/ThirdPersonController/API/index.html
- http://www.opsive.com/assets/ThirdPersonController/documentation.php
- https://medium.com/ironequal/unity-character-controller-vs-rigidbody-a1e243591483
- https://mariusgames.com/3d-character-controller-db7cd3a7b4df
- https://wiki.unity3d.com/index.php?title=CameraFacingBillboard