**Animation State Machines**

Each entity that can change to several animations will be bound using an animation state machine. This will allow us to control the entity’s visuals; locking their visual behaviour into the expected action they are currently performing.

**The Player:**

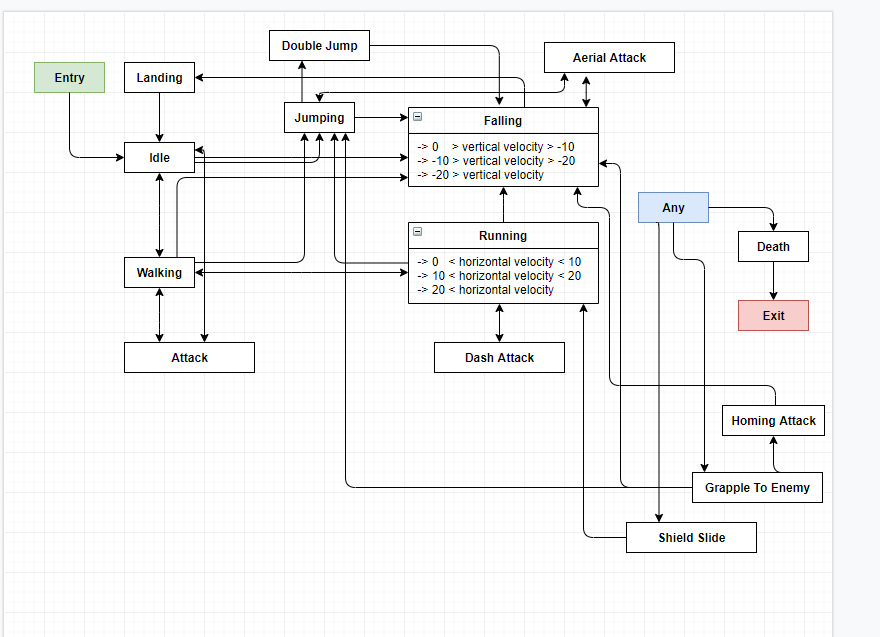


Figure 1- Player Character Animation State Machine.

**The Dizzy Charger:**

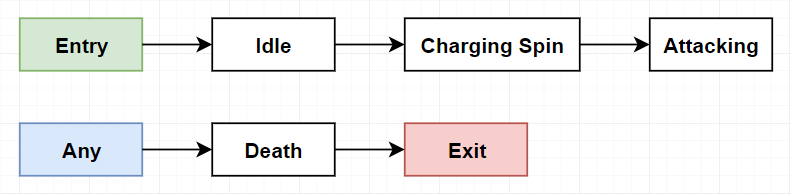


Figure 2 - Dizzy Charger Animation State Machine.

**The Domino:**

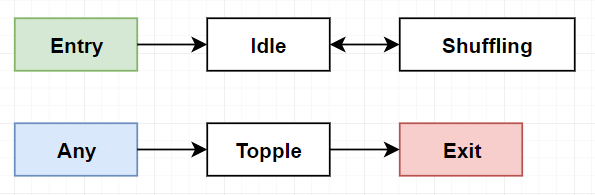


Figure 3 - Domino Animation State Machine.

**The Lobuzz:**

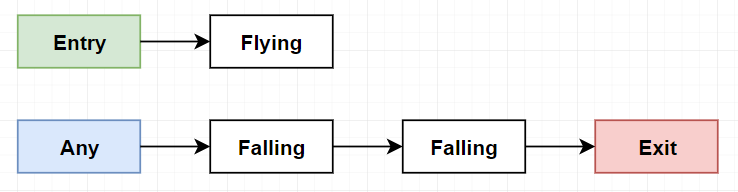


Figure 4 - Lobuzz Animation State Machine.

**The Shrew:**

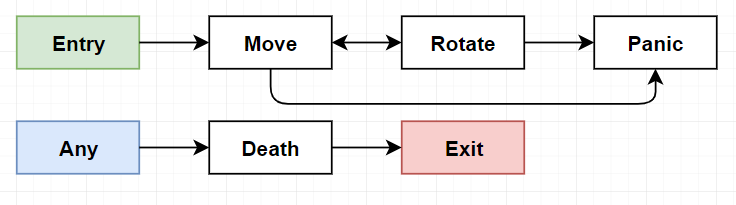


Figure 5 - Shrew Animation State Machine.

**The Catapult:**

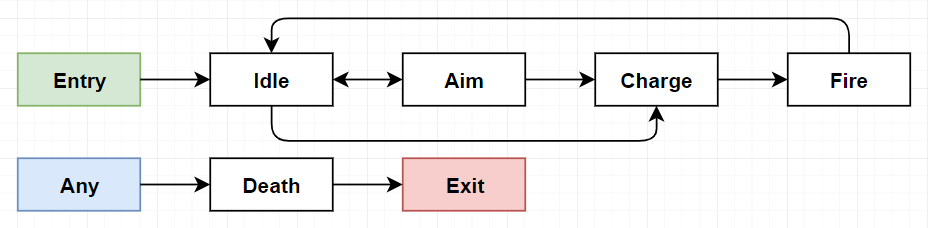
**

Figure 6 - Catapult Animation State Machine

**The Boomerang:**

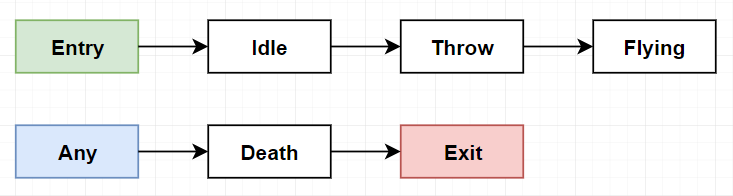
**

Figure 7 - Boomerang Animation State Machine

**Implementation**

We will utilise the Animation Blueprints in Unreal Engine 4.23. This will allow us to have to great control over the poses of the characters, but also great flexibility in the animations. The player character will handle most of the variables required, either by inherited values or by new ones added by us. An Enumeration will be added to keep track of the currently equipped weapon, as this will affect the active animation set. We will also make blend spaces between states to offer a smooth transition between animations, without the experience becoming jarring.

We took the liberty of implementing a basic Animation Blueprint for player in UE4 to visualise the process a bit easier.

A picture containing indoor, wall, bathroom

Description automatically generated

Figure 8 – Player state blueprint

This is the base state machine in the player character animation blueprint. The default state is having the sword equipped, and the other weapons are all linked through it. We would link them all together, but the way the conduits and handled (transitions) means this won’t be necessary.

A picture containing indoor, wall, bathroom, toilet

Description automatically generated

Figure 9- Sword and Shield Output

This is what is inside each of the nodes in Figure 8. It is literally just a nested state machine at this point, on which we will build up the animations.

A picture containing indoor, sitting

Description automatically generated

Figure 10 – Animation Transition

This is what the state machine holds. All these nodes are the different states the player can be in, whilst the selected weapon remains the sword and shield. The attack node will hold another state machine to choose the appropriate animation, but at this point, it will just play one animation for testing and debugging of the blueprint.

A picture containing indoor, wall

Description automatically generated

Figure 11 - Domino Animation State Machine.

This is inside the IdleToRun node. It holds a Blendspace, and a float controlling the interpolation between the different animations. This is very useful to us, because we can seamlessly transition between animations and provide a better user experience with the animations.

A screenshot of a computer

Description automatically generated

Figure 12- Event Graph

Finally, we have the event graph. This is where the blueprint gets the information to transition between states from. It can get variables from the player that we have defined such as the enumeration E\_Attacks to get the selected weapon, the Boolean Attacking, to see if we have pressed the attack key. However, it can also get variables held on the components of the player, such as the character movement holding the velocity and max walk speed of the player. We used this information to get a 0 to 1 value of the players speed, which works even if we change the max walk speed.