# Code design

Just a place to put all the stuff here

* State pattern for sword stances
* Forward declaring in header files
* Etc

# Animation design

* Need several idle/stance animations at each position of the sword, can then use blend space between them
  + E.g. sword in top left, sword at top, sword at top right
  + Blend space between topleft and sword at top dependant on mouse position
    - Blendspace is an asset like animation but an interpolation data
  + Code can be used to determine what quadrant the mouse curser is in and therefore what animation blendspace and blend parameter.
* State machine in blueprints
  + State Machines provide a graphical way to break the animation of a Skeletal Mesh into a series of States. These states are then governed by Transition Rules that control how to blend from one state to another. As a tool, they greatly simplify the design process for Skeletal Mesh animation, in that you can create a graph that easily controls how your characters can flow between types of animation without having to create a complex Blueprint network.

# Animation general setup

1. Create animation blueprint
2. Open animation blueprint and create state machine in AnimGraph
3. Connect state machine to output
4. Add a state to state machine (e.g idle/run)
   1. States could be SwordPositionTop/TopRight
   2. Add blend space to state or whatever (need to learn the possibilities)
5. Create a new C++ class which extends UAnimInstance
6. Reparent your blueprint to this class
   1. This class is the parent of your animation blueprint and is used to **extend** the functionality of the UAnimInstance. It is essentially a link between with extra functionality.
   2. Originally, all animation blueprints are derived from UAnimInstance
7. Once done, you can now access the functionality of the new C++ class in the animation blueprint.
   1. This includes variables, functions etc

* When you want a variable to control the switch between the states, you can use a blend space.
  + E.g. walking to running
* If you’re just switching states, then make a new state
  + E.g. running to jumping

## Accessing variables in AnimInstance in animation blueprint

* To access variables. The variable must be a UPROPERTY
  + E.g. UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "AvatarProperties")
* Object references can also be a UPROPERTY
  + Variables within the object reference can also be accessed if they are also made a UPROPERTY
  + E.g. having an avatar reference in AnimInstance can be made into a UPROPERTY and accessed.

## Switching animation states

When switching states with a condition, can have Boolean variables to denote which quadrant the mouse cursor is in and therefore easily switch/blend animations

These variables are calculated in the C++ code.

### Multiple state machines

* A state machine holds the animation state of the actor/avatar (e.g. walking, running etc) and can set the animation accordingly.
* For each Sword stance state, these animations are difference so separate state machines are necessary
* To switch between states, using “cached poses” connected to “blend poses by int” node, which is like a switch statement but for animation state machines.
  + The int is denoted in Avatar class.

### Root motion vs inplace with animation curves

* Better to drive the phsyics with animation curves as gives more control over the animation than root motion.
* Root motion disables input and has a fixed period (unless break up the animation manually?)
* Using animation curves should allow for changing animation and more control during the animations, e.g. can move with camera while dodging.
  + Extracting root motion using UE asset since does come with the asset pack..