# Code design

Just a place to put all the stuff here

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

## Speed up compilation times

* a dependency model that Unreal Engine follows and that C++ developers should follow in their Unreal projects to help speed up Visual Studio compilation. This model is called IWYU aka "Include What You Use”. The main four points of this model include minimizing the amount of include statements in your header files by using forward declarations and including CoreMinimal.h, including the matching .h files first in your .cpp files, not explicitly including precompiled header files, and avoid including monolithic header files such as Engine.h and UnrealEd.h. All of this and a few tips to help you enforce IWYU in your projects are covered.
  + <https://youtu.be/lQu7OCUku6I>

## Pointers or references in class variables

* “Recently, my mentor at work suggested that you should only use pointers when you plan on having the store point to a different object or null at some point. Instead, you should store references in classes.”
  + <https://stackoverflow.com/questions/37086638/should-i-store-references-in-classes-in-c/37087241#37087241>
* Benefit to reference, don’t need to mess around with nulls and limits undefined behaviour/users will need to use the class right.
* Cons, you now cannot copy assigned, move assigned, and the member cannot be changed (the reference cannot be rebound).
* However, there is nothing a pointer cant do that a reference can do, and is more versatile.
* Notes on dereferencing a pointer:
  + <https://stackoverflow.com/questions/4955198/what-does-dereferencing-a-pointer-mean>
* **Using pointers to store commonly used objects from variables such as Avatar.forwardvector such that you do not need to constantly fetch the reference/read the object from memory.** 
  + **When the object is required, the pointer is dereferenced.**
  + **Not sure what unreal engine will react when it cant copy the object sooooo.**

# Game design notes

* To slash or stab, hold down a button to activate slash and only activate slash at certain mouse speeds
* When slashing, allow for change of direction and change animation curve when this occurs
  + Only acknowledge change of direction if the mouse moves a certain threshold distance and different threshold angle.
  + Only allow for a few changes of direction, i.e. limit to 2 or 3.

# 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..

# Definitions

### Great channel

* <https://www.youtube.com/c/MathewWadsteinTutorials/videos>

### Great playlist

* <https://www.youtube.com/watch?v=OLQmeSsa9a0&list=PL9z3tc0RL6Z7U7U4hSu0epMmoh_dC3YFd>

### Forward and right vector

* Forward and right vectors are just directions. E.g. where its pointing.
  + <https://www.youtube.com/watch?v=M3wvhvrXrDI>
  + Can also get scene component forward vectors such as camera component

### Velocity

* Can get actor velocity with GetVelocity() which returns an FVector
  + https://www.youtube.com/watch?v=\_V1yLsn26vM
* This vector is in relation to the world axis.
* Max velocity is determined by the max walk speed in the character component (e.g. GetCharacterMovement();
* Can tell which direction its going in the world and what magnitude
* If called on a character, then will use the character component movement, if that’s not present it uses some physics system..
  + Get component velocity uses the physics system (not sure what this is tbh)
  + <https://www.youtube.com/watch?v=IYR-RgfJEHc>

### Unit direction vector

* Gets vector between one actor and another. Basically the direction to that actor
  + <https://www.youtube.com/watch?v=_JCWLsRyz0Y>

### Input movement

* Input movement:
  + AddMovementInput() and only applies to a chacter subclass of pawn
  + Can use Force to control when you can input on that axis
  + <https://www.youtube.com/watch?v=xkY8xqwmyFU>

### Add action/Axis mapping at runtime

* <https://www.youtube.com/watch?v=aiOVhsMBQ80>

### Get action/axis names

* <https://www.youtube.com/watch?v=tVrcdLADnjU>

### World transform

* Transform composed of Scale, Rotation (as a quaternion), and Translation.
  + Vectors and rotators
* Transforms can be used to convert from one space to another, for example by transforming positions and directions from local space to world space.
* Typically used to track an actor in space and move them
  + E.g. get world transform of the object, split it into components
  + Alter the component variable you want then setWorldTransform
  + <https://www.youtube.com/watch?v=48GW-LxKKlE> (set transform)
* <https://www.youtube.com/watch?v=j3V4Gji69GU>
* More info on world transform here
  + Sweep
  + Teleport etc
* GetRelativeTransform gets you the local transform of an actor
  + That is the relative to its parent (scene for some things.. idk, pretty sure root component too?)
  + I think it just gets you the local coordinate system of the object basically

### Debug tools in engine

* Debug arrow: <https://www.youtube.com/watch?v=JVwLZJ5ASuo>
* Debug plane: <https://www.youtube.com/watch?v=QdIdt73K3-Q>
* Debug point: <https://www.youtube.com/watch?v=_4LwI-BJj9c>
* Debug sphere: <https://www.youtube.com/watch?v=Jb9xAhBGxOM>
* Debug string: <https://www.youtube.com/watch?v=MGnEpP8ofFA>