OVERVIEW: PLAYER PACKAGE
MANY GAMES HAVE PLAYABLE CHARACTERS

- Playable characters can:
  - Build story and narrative
  - Provide a direct emotional hook for players
  - Provide a blank template for role-playing/power fantasy

- Regardless of intended experience:
  - Player spends the most time looking at them
  - Player interacts with the world through them
PLAYER PACKAGE

- General term for player character’s abilities and movements
- Directly tied to character animations and the responsiveness of the controls
- Coordinated effort by designers, programmers, and artists to create an enjoyable* way to interact with the game

*Or not...
CHARACTER MOVEMENTS

- Way in which a player moves the character through the world
  - Walking
  - Jumping
  - Flying
  - Swimming
  - Crawling
  - etc..
- Usually physically based
  - Simulated or kinematic
- Usually separate from “abilities” but not necessarily...
SIMULATED

- Object in the scene is subject to physics
  - Applied forces change its velocity
  - Interactions with other simulated objects affect it
KINEMATIC

- Object in the scene is subject to movements and trajectories outside of physics
  - Not necessarily subject to forces in the simulation
  - Affects simulated objects but not necessarily affected by them

Assassin’s Creed
SIMULATED OR KINEMATIC?

- Kinematic generally more common as baseline in player packages
  - Can still apply friction, air control etc
  - Reduces wild physics bugs
  - More designer control
- Kinematic objects can still be subject to physical forces
  - Usually handled via callbacks
- Can combine for a hybrid solution
**UE4: CHARACTER MOVEMENT COMPONENT**

- An Actor component that provides both movement functionality and *network replication* for movements
  - By default attached to Character Actors (a subclass of Pawns designed for bipedal playable characters)
- `PerformMovement` called during Tick to determine desired acceleration based on player input and settings
  - Once finalized calculations are made, movement is applied to the Character
- Movements sent to the server and applied authoritatively
CHARACTER MOVEMENT MODES

- Enum MovementMode provided to cover basic use-cases of character movement
  - **Walking** applies friction and allows “stepping up” but does not have vertical velocity
  - **Falling** applies gravity after stepping off an edge or jumping
  - **Flying** ignores the effects of gravity
  - **Swimming** applies gravity and buoyancy
  - **Custom** allows creation of custom functionality
CHARACTER MOVEMENT PROPERTIES

- Many, many parameters available for tuning movement
  - Basic physics concerns (mass, maximum acceleration, linear friction, gravity, etc)
  - Game-specific concerns (air control, ledge falling, client-server information, etc)
- If you have a question (how high can I jump, what is my max speed, have I landed, etc) there is probably a property that has an answer
- Helps to know some physics and networking terminology
CHARACTER MOVEMENT FUNCTIONS

- Multiple stages utilized for calculating the character’s movement

- Common functions to interact with CharacterMovementComponent within the Character/Pawn class are:
  - AddMovementInput
  - Jump
  - LaunchCharacter
  - Crouch/UnCrouch

- Common ways CharacterMovementComponent starts to process these are:
  - AddForce/AddImpulse
  - Crouch/UnCrouch
  - Launch
AN EXAMPLE: LAUNCHCHARACTER (CALLED FROM CHARACTER)

```cpp
if (CharacterMovement) {
    FVector FinalVel = LaunchVelocity;
    const FVector Velocity = GetVelocity();

    if (!bXYOverride) {
        FinalVel.X += Velocity.X;
        FinalVel.Y += Velocity.Y;
    }

    if (!bZOverride) {
        FinalVel.Z += Velocity.Z;
    }

    CharacterMovement->Launch(FinalVel);
    OnLaunched(LaunchVelocity, bXYOverride, bZOverride);
}
```
AN EXAMPLE: LAUNCHCHARACTER (HANDLED IN MOVEMENT)

void UCharacterMovementComponent::Launch(FVector const& LaunchVel) {
    if ((MovementMode != MOVE_None) && IsActive() && HasValidData()) {
        PendingLaunchVelocity = LaunchVel;
    }
}

bool UCharacterMovementComponent::HandlePendingLaunch() {
    if (!PendingLaunchVelocity.IsZero() && HasValidData()) {
        Velocity = PendingLaunchVelocity;
        SetMovementMode(MOVE_Falling);
        PendingLaunchVelocity = FVector::ZeroVector;
        bForceNextFloorCheck = true;
        return true;
    }
    return false;
}
CHARACTER INTERACTIONS

- Way in which the playable character interacts with the world and other playable and non-playable characters
  - Fighting
  - Building
  - Puzzle-solving
  - Talking
  - etc...

- Implementation depends heavily on the game
HIT AND HURT BOXES

- Primarily terms in fighting games, but used in any game where player characters can deal or receive damage
  - Can more generally be called collision volumes
- Hit boxes provide event information for when the player character has hit something
- Hurt boxes provide event information for when the player character has been hit by something
NOT STRICTLY WRONG...

Lag compensation in Counter Strike

Skullgirls

Guilty Gear
IMPLEMENTATION AND DESIGN

- The combinations of hit and hurt (plus additional things like block proximity) boxes leads to a lot of potential states in fighting games
  - Concepts like fuzzy guard and option selects come from these edge cases*

*the first fighting game "combo" system was a bug in Street Fighter II

https://www.youtube.com/watch?v=jdGO2rfeKrQ
PICKUPS AND DROPS

- Ability to equip and unequip items or weapons
  - May or may not involve an inventory
- Item is a separate actor
  - Memory management separate from player character
- Location and orientation matches player character

Spec Ops: The Line
ATTACHING AND DETACHING OBJECTS IN UE4

- Two ways to attach an actor to another actor:
  - AttachToActor
    - Attaches to root component of Actor
  - AttachActorToComponent
    - Attaches to specified component of Actor

- Both functions will work in most situations and both can specify a named socket (e.g. attach an item to a character’s hand etc) to attach to

- FAttachmentTransformRules specifies how the attached Actor should move relative to the parent Actor
UE4: OBJECT SPAWNING

- Can spawn Actors using `UWorld::SpawnActor()`
  - Creates a new instance of specified class
  - Returns pointer to that object
  - Specifies initial position and orientation of spawned Actor

- Can spawn Blueprint Actors by using `FObjectFinder` to retrieve the UClass based on the BP Reference path then calling `SpawnActor` as usual:

  ```cpp
  MyActor * spawnedActor = GetWorld()->SpawnActor<MyActor>(MyActorBP, FVector::ZeroVector, FRotator::ZeroRotator);
  ```

  Note: Keep a reference to the spawned object if you want to remove it later
CONTEXT-SENSITIVE ABILITIES

- Abilities that are only available during certain times under certain conditions
  
  - Can implement using a combination of raycasts/trigger volumes and character state to determine how to interact
CHARACTER STATE

- Games are inherently very stateful
  - Awful for programming but it’s what makes them engaging and dynamic

-Playable characters tend to have many different states as well
  - Idling, Walking, Running, Jumping, Dashing, Crouching, Diving, Interacting, Striking, On Cooldown, Taking Damage, Injured, Dying, Dead, etc...
  - Note that these states are not necessarily exclusive...
WHAT WE NEED TO KNOW...

▸ Can I transition from my current state to the requested state?

▸ How do I update my state?
Finite State Machines (FSM)

- Mathematical model of computation that describes a collection of states the machine can be in at any time
  - Must be in exactly one state
  - Can only transition between certain states

Wikipedia example: coin-operated turnstile
Animation states for a character

Possible to implement FSMs in a number of different ways (not implemented by default in UE4 outside of animations)

Regardless of implementation, useful to connect to UENUMs so that enumerated states are exposed to Blueprints and AnimationBlueprints (more on that later)

Also regardless of implementation, thinking through and careful planning of your states and transitions is essential for avoiding corner case bugs (which will happen regardless)
FURTHER READING

- [frametrapped.com](https://frametrapped.com/)
- [gamecrate.com/how-e-sports-understanding-hitbox-meme/16773](https://gamecrate.com/how-e-sports-understanding-hitbox-meme/16773)