* **PlayState.hx**
  + Initializes the cameras used
    - utilities/camera/Camera.hx is a wrapper class for FlxCamera.
  + Create background (FlxSprite)
  + Create teams
    - team/Team.hx
  + Create players and give them a controller
    - player/Player.hx
    - utilities/controller/Controller.hx
  + Each player is given a battle menu. Each menu is assigned a position on screen.
    - player/menu/BattleMenu.hx
  + Create player characters. Each character is a super class of the player character class.
    - player/PlayerCharacter.hx
    - player/playerCharacterClass/Wizard.hx
    - player/playerCharacterClass/Paladin.hx
    - player/playerCharacterClass/ Fighter.hx
    - player/playerCharacterClass/ Ranger.hx
  + Create monster group RhinoGroup.hx
    - monsters/MonsterGroup.hx
    - monsters/monsterGroup/RhinoGroup.hx
* **library/Library.hx** – Enum of classes used by all elements of the project
  + library/Players.hx
    - Tracks all players
    - Each player created is given a unique ID (0-99)
  + library/Controllers.hx
    - Tracks all controllers
  + library/Character.hx
    - Tracks all characters
    - Give each character a unique ID (1000-9999)
    - Can retrieve arrays of other active characters by team:
      * self, all, allies, alliesExcludingSelf, others, enemies
        + Each function requires the caller to supply their self as an argument
        + All characters are enemies of characters whose team is null
        + Each of the above functions has an optional argument to include dead characters
      * deadAllies, deadOthers, deadEnemies
        + Each of these has a placeholder argument to ensure they have the same signature as the normal functions
        + Currently monster characters are instantly deleted when they die
  + library/Teams.hx
    - Track all teams
    - Give each team a unique id (100-999
  + library/CardOverseer.hx
    - Track all cards by owner’s character ID
    - Give each card a unique ID (10000-infinity)
  + library/UniqueIDBot.hx
    - For all other objects that need a unique ID
    - Gives ID’s from -1 to negative infinity
  + library/Cameras.hx
    - Enum of cameras used by game
  + library/ElementalResistances.hx
    - Enum of damage resistances/vulnerabilities by element type
    - Get function (takes DamageTypes, returns Resistances)
  + library/StatTables.hx
    - Table of ability modifiers by stat level
  + library/DamageColors.hx
    - Lookup for color by DamageTypes
  + library/StatusColors.hx
    - Lookup for color by StatusTypes
  + library/CardColors.hx
    - Lookup for color by CardFamily
* **utilities/plusInterface**
  + An interface that extends functionality built into FlxSprite, FlxBasic, and FlxState
  + Each plus object has a parent and tracks sub objects
  + Destroying a plus object also destroys all sub plus objects
  + Provides an alternate update method (plusUpdate) which can allows objects to pause updating of an object and all of its sub objects
    - These object’s normal update function is still called
    - This is useful for delay/freeze/stop status effects and more
    - plusUpdate updates the leaf objects first and root last
    - objects have a bool forceUpdate that overrides this
  + Each plus interface object requires a parent object at creation
    - The parent tracks the new object
    - The newly created object tracks its parent
  + PlusInterface.hx - Interface object
    - private var plusType:PlusEnum;
    - public function type():PlusEnum;
    - private var trackedItems:Array<PlusInterface>;
    - private var tracker:PlusInterface;
    - public function addItem(item:PlusInterface):Void;
    - public function setTracker(tracker:Null<PlusInterface> = null):Void;
    - public function removeItem(item:PlusInterface):Bool;
    - private var updateEnabled:Bool;
    - public var forceUpdate:Bool;
    - public function setUpdate(enabled:Bool):Void;
    - private function plusUpdateSubroutine(elapsed:Float):Void;
    - public function plusUpdate(elapsed:Float):Void;
    - public function destroy():Void;
  + PlusEnum.hx
    - Enum of available plus interface objects
    - Used to by plusType to determine object’s type
  + BasicPlus.hx
    - extends FlxBasic and implements PlusInterface
  + BasicPlusTracker.hx
    - implements PlusInterface and intended to be used much like an array of BasicPlus objects
  + SpritePlus.hx
    - extends FlxSprite and implements PlusInterface
  + SpritePlusTracker.hx
    - implements PlusInterface and intended to be used much like an array of SpritePlus objects
  + StatePlus.hx
    - extends FlxState and implements PlusInterface
    - StatePlus does not take a parent object on creation
      * Intended to act as the root for objects in the state
      * Has the ability to disable all plusUpdate calls for all objects in state
        + includes objects where forceUpdate=true
  + TimerPlus
    - Extends BasicPlus
    - A timer that calls a TimerPlus->Void function when the timer ends
      * Uses itself as an argument for the TimerPlus argument
      * Destroy itself when call completed
    - Makes the function call first if it would be destroyed
* **utilities**
  + utilities/animation/Animation.hx – wrapper for FlxSprite.Animation
    - extends SpritePlus
    - Takes arguments for the number of frames in the animation and the runtime of the animation
    - Optional argument for the number of times the animation loops (will play for a total of loops\*runtime seconds)
    - play and stop functions
  + utilities/button/Button.hx
    - Extends SpritePlus
    - Designed to be used with a mouse. Many objects extend this class but mouse functionality isn’t being used
  + utilities/camera/Camera.hx
    - Extends FlxBasic. Wrapper class for FlxCamera
    - Mostly a holdover from old Far Horizons project. Mostly functions like a FlxCamera
    - function flxCam returns FlxCamera object
  + utilities/controller/Controller.hx
    - Acquires the first available active and unclaimed FlxGamepad
    - Each button has an integer tracking how long it’s been held down or how long it’s been released in number of frames
      * 1 = button just pressed
      * >2 = button held for 2 or more frames
      * 0 = button just released
      * <-1 = button released for 2 or more frames
  + utilities/event
    - Classes and enums used for mouse events (not used)
  + utilities/observer
    - Observer and subject classes. Originally used with mouse interface in Far Horizons (not used now)
  + utilities/Selection
    - (TODO move to player folder?)
    - A selection of buttons
    - Used to track possible options that the player (Player.hx) can choose from
    - Player class provides controller inputs
    - Tracks Void->Void functions that will be called when a given button is pressed
      * Generally these functions iterate through the selection and activate functions like playing a card
    - Player class has a base selection. This selection can have sub selections. When calling a function in a selection, the super selection will pass the call down to the lowest sub selection.
      * This makes selections act like stacks
        + Selections can be pushed onto the stack and popped off
    - Can have multiple “slices”. Generally this is used when playing a card that primarily targets enemies but can target allies (or vice versa)
* **player**
  + Player.hx
    - Extends SpritePlus
    - Visually represented by a cursor
    - Moves between various buttons (its target)
    - Has a BattleMenu, Selection, PlayerCharacter, and Controller
    - Its base Selection is provided by the BattleMenu
    - function cardFinished(card:PlayerCard)
      * discards the card and resets focus for the current selection
  + player/menu
    - BattleMenu.hx
      * Contains MenuButtons, Selection, healthBar, staminaBar, manaBar, and guardText
      * Contains functions for interacting with BattleMenu through its selection
      * Updates healthBar, staminaBar, manaBar, and guardText with values from its owner’s (player’s) character
    - MenuButton.hx
      * Refrences a card in the player’s hand
      * Text and color change based on card
  + PlayerCharacter.hx
    - Extends Character
    - Function for adding card to deck
    - PlayerCharacters draw and discard cards differently than MonsterCharacters
      * When a player discards a card the card slot is replaced with a temporary unplayable card called Redraw. Only players use Redraw
      * One a Redraw charges it has the player remove it and replace it with a card from the player’s deck. (This uses the function drawPlayerCard)
      * If a card in the player’s hand is null, then the player will replace the card with a Redraw
      * If the player has too many cards in hand then extra cards are discarded (function removeCardSlot)
    - When a player plays a card, it resets all cards in hand that match the played card’s CardFamily
    - Ability to kill and revive PlayerCharacters
    - override updateStats: update the menu when stats change (change number of buttons in selection when hand size changes)
  + player/playerCharacterClass
    - Fighter, Paladin, Ranger, and Wizard character classes
  + player/playerCards
    - cards used by players sorted by CardFamily
    - PlayerCard.hx
      * player:Player – the player that owns the character that owns the card
      * targets:Array<Array<Character>> - array of possible targets
      * override public function play()
        + Check to make sure targets isn’t empty and exclude invalid targets
        + return if player doesn’t have the necessary resources
        + Add a sub Selection of targets to the Player
      * PlayerCard has different functions than BattleMenu for navigating through selection
      * ok function calls Card function beginResolution
      * esc pops current PlayerCard Selection
      * fail()if the selection became empty (should be called if the card has not started resolving but the card can’t be played)
        + resets player cursor to the BattleMenu Selection
      * finish() called when the card is finished
        + resets player cursor to the BattleMenu Selection
        + call player.cardFinished()

discards the card and resets focus for the current selection

* + - * override public function beginResolution()
        + sets target to the chosen Selection item
        + calls Card’s beginResolution()
* **character**
  + Character.hx
    - Has:
      * discard Deck and activeCards (hand) Deck
      * statusEffects
        + effects that all have a simple implementation (none of them stack and they all use a timer)
      * effects
        + Like statusEffects but for effects that require a more complex implementation
      * states
        + flying, distanced, etc…
      * resistances
      * damageBonus
      * takesDamageCalls
        + An array of arrays of damage calls
        + used to inset additional functions into the takesDamage process
        + (used by Defend PlayerCard)
      * doesDamageCalls
        + An array of arrays of damage calls
        + used to inset additional functions into the doesDamage process
      * damageNumbers
        + The numbers and text that appear above characters
      * stats
        + handSize, speed, maxHealth, etc…
      * levels
        + Character’s ability score levels
      * onDeckCard
        + Only used by monsters. The card that is about to be played.
      * windupCard
        + Card being played that is in the “windup” phase
      * resolvingCard
        + Card being played that is in the resolving phase (used by CounterAttack)
      * positionX and positionY
        + Base position. May not be the same as sprite’s position (for example, the sprite moves 100 pixels when the character is distanced)
    - giveResources(resources:Resources)
      * adds resources to Character.resources
      * triggers DamageNumbers for each resource added
    - removeResources(resources:Resources)
      * similar to giveResources
    - giveStatusEffect(type:StatusTypes,value:Float)
      * value refers to the time the effect will last
    - takesDamage(types:Array<DamageTypes>, value:Float, ?cardType:Null<CardType> = null, ?source:Null<Character> = null):Float
      * Allows various functions to modify the value of the damage taken or cause additional effects
        + takesDamageCalls[0]
        + effects (effects has 2 different takesDamage calls)
        + resistances
        + stats (damage resistance stat)
        + statusEffects
        + set damage to a minimum of 1
        + effects (guard effect must come after other calls)
        + takesDamageCalls[1]
      * If damage taken is >0 but <1 then set it to 1
      * The character takes the damage and triggers DamageNumbers
      * returns the final damage taken
    - doesDamage(types:Array<DamageTypes>, value:Float, ?cardType:Null<CardType> = null, ?source:Null<Character> = null):Float
      * Allows various functions to modify the value of the damage done or cause additional effects
        + doesDamageCalls[0]
        + statusEffects
        + stats
        + damageBonus
        + doesDamageCalls[1]
      * Returns the final damage done
    - setTeam and getTeam
      * setTeam removes character from old team
    - targetThis (source:Card):Character
      * Checks effects to see if any trigger by being targeted (for example Blessing and Curse)
    - canBeTargetedBy and canBeTargetedByType
      * Checks if this Character is a legal target for the Card or CardType (returns bool)
    - drawCard():Bool checks to see if card can be drawn
      * This check is used by both PlayerCharacter and MonsterCharacter
    - addCard adds a new Card to the Character’s Deck
    - cardsInHand():Array<Card>
      * returns a list of the cards currently in the player’s hand
    - updateStats()
      * must be called after levels change
  + deck/Deck.hx
    - works much like an array
      * has push and pop functions
    - has known and unknown parts
      * knownCards is an Array<Card> and can be publicly accessed
      * At initiation, the number of known cards (numKnown) is set
      * Cards are moved from unkown to known if len(knownCards) < numKnown
      * Moving cards from unknown to know uses the private function popUnknown()
      * popUnknown has a 50% chance of drawing the bottom card of unknownCards, a 25% chance of drawing the second, 17.5% of the third, etc…
  + damage/DamageTypes.hx
    - Enum of DamageTypes
      * physical, magic, fire, cold, water, earth, nature, poison, lightning, holy, dark, psychic
  + damageNumbers
    - DamageNumbers.hx
      * Triggers damage numbers (or text) to appear above Characters
    - DamageNumbers.hx
      * The actual number (or text) that appears on screen
      * Deletes itself when it is finished
  + damageBonus/DamageBonus.hx
    - stores a map of bonuses (multipliers)
    - Has its own default map but can store maps of modifiers from other sources
      * Stored by source’s ID
      * These modifiers can be updated or removed later
    - When the Character doesDamage, DamageBonus.doesDamage modifies the damage value
      * If the bonusMap for the DamageType is >0 then value = value \* (1 + bonusMap[type] / 100 / types.length)
      * else value = value / (1 - bonusMap[type] / 100 / types.length)
  + resistances/Resistances
    - Functions much like DamageBonus except:
      * If the resistanceMap for the DamageType is >0 then value = value / (1 + resistanceMap [type] / 100 / types.length)
      * else value = value \* (1 - resistanceMap [type] / 100 / types.length)
  + statusEffects
    - StatusEffects.hx
      * Map of timers for status effects
        + each value is the time remaining for the effect
      * StatusEffects don’t stack (if (value > statusMap[type]) statusMap[type] = value;)
      * Cold increases Ice damage done and taken
        + Cold reduces Fire damage done and taken
      * Hot increases Fire damage done and taken
        + Hot reduces Cold damage done and taken
      * Cold and Hot cancel each other out
      * Wet and Hot cause each other to run out faster
        + Wet increases the duration of Cold
        + Wet increases Lightning damage taken
        + Wet causes a character to take Lightning damage when dealing lightning damage
      * Rage causes the owner to deal extra damage with physical melee attacks
        + Rage causes reduced damage from physical attacks
        + Rage causes increased damage from psychic attacks
      * Drugged reduces draw speed (Change name?)
      * Slow reduces Card charge speed
      * Delay causes character to pause most (generally excludes statusEffects and Effects) timers for a short time
    - StatusTypes.hx
      * Enum of StatusEffects
  + effects
    - Effects.hx
      * Similar to StatusEffects but used for more complex effects
      * Venom does damage over time
        + X damage every second where X is the number of Venom stacks
        + If Venom has a timer and a number of stacks
        + If the timer runs out then there is a 1 second window to apply more Venom before the count resets to 0
      * Combo is used by some cards to keep track of the character’s combo multiplier
      * Defender is used for one character to intercept damage for another
        + Adds a call to Character.takesDamageCalls
        + Defending player takes full damage (reduced by Resistances and Guard)
        + Defended player takes damage equal to what Defending player takes after reduction (which is then reduced by Defended player’s Resistances and Guard)
      * Barrier acts like temporary health and blocks damage
        + Barrier takes damage before checking Resistances or Guard
      * Guard reduces damage taken but taking damage removes guard
        + damagePrevented =value - value /(Guard/20 +1) up to a max of Guard and down to a min of 1
        + Guard -= damagePrevented
        + value -= damagePrevented
      * Blessing prevents a negative magic effect
      * Curse prevents a positive magic effect
    - Defender.hx
      * Used for instances of defender
      * Tracks time the instance lasts, source (Character) of the instance, and the Card function to be called when defended Character is damaged
    - Guard.hx
      * used to track individual instance of Guard
  + states/States.hx
    - Set of Booleans:
      * distanced – can only be hit by ranged attacks, charge attacks, or melee attacks by flying characters
      * flying - can only be hit by ranged attacks, or melee attacks by flying characters
      * combos – character will track its combo (Effect) score
  + stats
    - Levels.hx
      * Character’s level in each ability type
      * Has its own default map but can store maps of modifiers from other sources
        + Stored by source’s ID
        + These modifiers can be updated or removed later
    - LevelsEnum.hx
      * strength, dexterity, endurace, constitution, resilience, intelligence, wisdom, willpower, insight
    - Stats.hx
      * Tracks stats
      * Stats are based on level and can be modified by other sources
        + Stored by source’s ID
        + These modifiers can be updated or removed later
      * bonusDmg (ToDo change name) causes physical attacks to do more damage
      * magicDmg causes magic attacks to do more damage (note: attacks could be both magic and melee. SpellBlade for example)
      * dmgResistance reduces all damage taken
      * handSize changes the character’s hand size
      * speed effects how fast cards charge
      * drawSpeed effects how fast cards are drawn
      * maxHealth is the characters maximum health
      * stamina (change to maxStamina) is the character’s maximum stamina
      * stamina Regen is the time it takes to regenerate a point of stamina
      * mana (change to maxMana) is the character’s maximum mana
      * manaRegen is the time it takes to regenerate a point of mana
      * perception (to do something at some point maybe)
    - StatsEnum.hx
      * Enum of stats
* **monster**
  + MonsterCharacter.hx
    - target:Null<Character> - The target of the monster’s next card
    - defaultAction:MonsterCard – The card the monster plays if it can’t play any others
    - delayTime:Float - The time before the monster chooses its next action
      * = delayBaseTime + delayRandomExtraTime \* random float
    - delayBaseTime:Float – Base delay time
    - delayRandomExtraTime:Float – Random extra delay time
    - highlight:SpritePlus – Used to show what the monster is targeting
    - chooseAction():Card
      * Picks a card to play and a target for the card
      * For each card in the monster’s hand, it chooses a random card to play
        + (ie. With 4 cards in hand it could choose 1,2,1,0)
      * If a card can’t be played, then it tries the next one
      * If none of the randomly chosen cards can be played, then it plays its defaultAction Card instead
    - discardCard(card:Card)
      * discards the played (or sometimes unplayed) card and resets delayTime
    - plusUpdate(elapsed:Float)
      * Triggers chooseAction
      * Triggers playing the card when it is ready
      * Manages highlight
  + MonsterCard.hx
    - possibleTargetsFunct:Character->Bool->Array<Character>
      * Set in each MonsterCard. Determines the possible targets a monster can pick from.
        + (for example: possibleTargetsFunct = Library.characters.enemies)
    - getTarget():Null<Character>
      * Returns a random valid target from possibleTargetsFunct
      * If there are no valid targets, then it return null
    - play()
      * Gets target from monster
      * Begins resolving the card
  + monsterGroup/MonsterCard.hx
    - A set of monsters which get summoned together
    - addMonster(monster:MonsterCharacter, position:Float)
      * Add a monster to the group
    - setTeam(team:Team)
      * Sets the group’s team
    - dead():Bool
      * Returns true if all monsters in group are dead
* **card**
  + Card.hx
    - owner:Character – owner of the card
    - enabled:Bool – whether or not the card is charging
      * should be enabled in hand and disabled in deck
    - name:String – cards name
    - cost:Resources – cost in resources
    - family:CardFamily – the cards family (color)
    - cardType:CardType – types such as melee, ranged, magic, spell, positiveEffect etc…
    - elements:Array<DamageTypes> - array of elements, can be empty
      * (effects resistances granted to owner)
    - resistances:Resistances – resistances granted to owner
    - Cards have up to 3 phases
      * Charging – cards can be played when charged
      * Windup – short delay before card gets played
      * Resolving – some cards resolve over a short period of time, others resolve instantly
    - charge:Float – amount the card has charged
    - chargeTime:Float – amount of time it takes for the card to charge
    - isCharged:Bool – if the card has finished charging
    - windup:Float – how far the card is into its windup phase
    - windupTime:Float – time the windup phase lasts
      * A value of 0 will skip this phase
    - windupAnimation:Null<Animation> - the animation played during the windup phase
    - isResolving:Bool – if the card is in the resolving phase (needed by monster?)
    - resolveAnimation:Null<Animation> - animation played when the card resolves
    - stacks:Int – some cards gain additional stacks after charged
      * Could be used to give a card extra functionality if it is allowed to charge even longer
    - isStackable:Bool – if the card gains stacks
    - target:Null<Character> - The cards target. Must be set to resolve
    - discardActions:Array<Card->Void> - not used. May need to be removed
    - resolveDelay:Float – how far the card is into its delayed resolve phase
    - resolveDelayTime:Float – how long the delayed resolve phase lasts
      * A value of 0 will skip this phase
    - resolveDelayDamageTrigger:Null<Array<DamageTypes>->Float->Null<CardType>->Null<Character>->Float>
      * a damage trigger that can interrupt or trigger a card with a delayed resolve
      * CounterAttack uses this
    - resolveDelayDamageContainer:Null<Array<Array<DamageTypes>->Float->Null<CardType>->Null<Character>->Float>>
      * the array that the trigger got added to
      * so it can be removed when done
    - resolveDelayAnimation:Null<Animation> - animation played when card resolves after the delayed resolve phase
    - delayAnimation:Null<Animation> - animation played during delayed resolve phase
    - fizzle()
      * Called when the card is CounterSpelled or Parried
    - finish()
      * Called when the card is finished resolving
      * Removes damage triggers
      * owner.windupCard = null;
      * owner.resolvingCard = null;
      * calls discard()
    - discard()
      * calls discardActions
      * calls owner.discardCard(this)
      * calls resetCard()
    - resetCard()
      * resets card timers and Booleans to their default values so the card is ready to be played again
    - resetCharge()
      * resets the cards charge (charge = 0 and isCharged = false)
    - play()
      * Empty function in base class. Implemented in PlayerCard and MonsterCard
    - beginResolution():Bool
      * Attempts to start resolving the card
      * Returns false if it can’t. This occurs under the fallowing conditions:
        + target == null
        + owner does not have the required resources
      * Plays windup animation
      * isResolving = true;
      * If card uses windup phase
        + owner.windupCard = this;
        + give owner the Delayed StatusEffect for the time the windup phase lasts
      * else
        + begin resolving card
    - resolve()
      * Double check the card hasn’t been countered
        + enabled will == false if it has
      * If card uses delayed resolve
        + owner.resolvingCard = this;
        + owner.windupCard = null;
        + Play resolve delay animation
        + Set damage triggers if they are used (if they != null)
      * else
        + play resolve animation and call finish()
    - delayedResolve()
      * Called when delayedResolve phase finishes
      * Playes resolveDelayAnimation
      * Calls finish()
    - damageTrigger()
      * Placeholder. Overrided by functions that use it
    - damage()
      * function to have the card deal damage to the target
    - setDamageTrigger
      * Sets up the damage trigger
      * Makes sure the function is added to the intended array only once
    - plusUpdate()
      * Handles charging and stacking
      * These functions are paused when the owner is delayed
    - update()
      * Handles windup and delayedResolve
      * These functions are NOT paused when the owner is delayed
  + CardFamily.hx
    - An enum of the card families
    - Red, Blue, Green, Yellow
      * The basic colors
      * Playing a card of these colors resets all cards that share their color
    - RedBluec, RedGreen, RedYellow, BlueGreen, BlueYellow, GreenYellow
      * Combinations of the basic colors
      * Playing a card of these colors resets all cards that share all of their colors
      * They never reset each other
    - White
      * Is all colors except black
    - Black
      * Only shares a color with itself
    - Equipment
      * Doesn’t share a color with any other card including other equipment cards
    - Unplayable
      * Cards that are not to be played (ex. Redraw)
  + CardType.hx
    - melee
      * If the card is melee range
      * excludes ranged
    - ranged
      * If the card is ranged
      * excludes melee
    - spell
      * If the card is a spell
      * Sets magic == true if spell == true
      * Sets ranged == true if spell == true
      * Note: a spell can be melee.
        + First set spell(true) then melee(true)
    - magic
      * if magic == false then spell == false
    - piercing
      * the only type that uses a float
      * uses a value from 0 to 1
      * The amount that the damage ignores Guard
    - trueDamage
      * same functionality as piercing(1)
    - charge (rename?)
      * Charge attacks are generally melee attacks that can close the distance on a distanced character.
      * The attacked character is meant to lose distanced.
        + However this must be implemented in the Card. Charge doesn’t automatically effect the attacked character’s state
      * positiveEffect
        + A positive effect (Checked by Blessing and Curse)
      * finesse
        + Not used. may be removed
      * effectsLiving
        + Unimplemented
      * effectsDead
        + Unimplemented