Entities & Responsibilities

GameWorld: Manages the game instance & the interface between it and the players (e.g processing for events) by running the main game loop

Tile: Represents the tiles which the dragons will interact with (stand on)

DrawProperties: Data class for organising the drawing properties required for drawing any element

PlayableCharacter: Represents the playable character a player interacts with

GameBoard: Represents the game board. It initialises the game board & runs the interactions with the game board by the players (e.g performing movement, flipping chit card)

ChitCard: Represents the chit cards and their effects

EventBus: Handles registration of listeners, and notification of appropriate listeners on event fire. [Class is for organising all EventListeners into one place]

WinEventXxxx....: Publisher and listeners for win event

MoveActionXxxx....: Publisher and listeners for a move action (for characters) that is fired

DrawableByAsset: Indicates that the object is drawable by pygame using assets

DrawAssetInstruction: A data class for organising data required for drawing an asset

ModularClickableSprite: Allows classes to be represented as a sprite that is clickable on a screen.

PlayableCharacterVariant: Represents a variant for the playable character entity

PygameScreenController: Contains useful methods for interacting with pygame's (the game's) screen

main.py: Game config before start, and serves as entry point to game

Patterns Used

Observer: WinEventPublisher, WinEventListener
• Why?: Don't have to check all starting tiles to see if win occured. Allows for wins from other sources

Singleton: EventBus, PygameScreenController
• Why?: Should be one central event bus managing all events

Todo

Cardinalities
PygameScreenController

Later Refactorings Required

Tile now can retrieve & store animal (optional?). Animals are crucial to the movement system of the game.

onClick now takes in PlayableCharacter and Tile

<<enum>>
Animal

.salamander
.baby_dragon
.spider
.bat

AnimalChitCard

- animal: Animal

+ __init__(Int, DrawProperties = None)
+ get_draw_clickable_assets_instructions() : [(DrawAssetInstruction, Clickable)]
+ on_click(PlayableCharacter, Tile) : Void

PirateChitCard

+ __init__(Int, DrawProperties = None)
+ get_draw_clickable_assets_instructions() : [(DrawAssetInstruction, Clickable)]
+ on_click(PlayableCharacter, Tile) : Void

<<enum>>
PlayableCharacterVariant

.blue
.green
.orange
.purple

DefaultGameBoard.__init__: First parameter is the sequence of tiles as their identifiers. Second parameter is [(starting tile, tile # to connect to in first parameter, playable character to start in it)]. The second parameter (starting tiles) will be used to make insertions to main tile sequence property. Third parameter is the chit cards to use for the DefaultGameBoard.
DefaultGameBoard.starting_tiles: [starting tiles]
DefaultGameBoard.move_character_by_steps: Moves a dragon from its tile by x number of steps
DefaultGameBoard.flip_chit_card: Flip a chit card on this game board. Chit card object will be obtained by click listener in pygame.
DefaultGameBoard.add_chit_card: Add a chit card to the game board

DefaultGameBoard

- main_tile_sequence: [Tile]
- starting_tiles: [Tile]
- chit_cards: [ChitCard] = []

+ __init__([Tile], [(Tile, Tile)], [ChitCard])
+ move_character_by_steps(PlayableCharacter, Int) : Void
+ get_character_floor_tile(PlayableCharacter) : Tile
+ flip_chit_card(PlayableCharacter, ChitCard) : Void
+ add_chit_card(ChitCard)
+ get_draw_assets_instructions() : [DrawAssetInstruction]
+ get_draw_clickable_assets_instructions() : [(DrawAssetInstruction, Clickable)]

ChitCard.__init__: Create a chit card with the # number of symbols

ChitCard.on_click: Perform the chit card's effect on the dragon based on the tile it is on. [don't compose dragon with tile (+ itll cause cicrcular dependency) because its only necessary for chit card's effects] . Called from GameWorld

ChiCard.set_flipped: Set flippeds state

<<abstract>>
ChitCard

- flipped: Bool
- symbol_count: Int
- draw_properties: DrawProperties

+ __init__(Int, DrawProperties = None)
+ get_draw_clickable_assets_instructions() : [(DrawAssetInstruction, Clickable)]
+ on_click(PlayableCharacter, Tile) : Void
+ set_draw_properties(DrawProperties) : Void
+ get_draw_properties(): DrawProperties
+ set_flipped(Bool) : Void

ModularClickableSprite.get_draw_clickable_assets_instructions(): Get the instructions to draw the clickable assets. In form (draw instruction, clickable object)
ModularClickableSprite.on_click: Allows a class to respond to a click. PlayableCharacter and the Tile its on is passed in

<<interface>>
ModularClickableSprite

+ on_click(PlayableCharacter, Tile) : Void
+ get_draw_clickable_assets_instructions() : [(DrawAssetInstruction, Clickable)]

DrawAssetInstruction

- asset_path: String
- x: Int
- y: Int
- size: (Int, Int)
- rotate: Float

+ __init__(String, Int, Int, (Int, Int), Float)
+ get_asset_path(): String
+ get_x_coord(): Int
+ get_y_coord(): Int
+ get_width(): Int
+ get_height(): Int
+ get_rotation(): Float

<<interface>>
DrawableByAsset

+ get_draw_assets_instructions() : [DrawAssetInstruction]

DrawableByAsset.get_draw_assets_instructions: Gets the drawing instructions for drawing an object onto the pygame screen using assets

PygameScreenContoller

+ __init__([PlayableCharacter], [ChitCard], GameBoard)
+ run(params): Void
+ on_player_win(PlayableCharacter) : Void

GameWorld

- playable_characters: [PlayableCharacter]
- game_board: GameBoard

+ __init__([PlayableCharacter], GameBoard)
+ run(params): Void
+ on_player_win(PlayableCharacter) : Void

1..1

<<abstract>>
GameBoard

+ move_character_by_steps(PlayableCharacter, Int) : Void
+ get_character_floor_tile(PlayableCharacter) : Tile
+ flip_chit_card(PlayableCharacter, ChitCard) : Void
+ add_chit_card(ChitCard): Void
+ get_draw_assets_instructions() : [DrawAssetInstruction]
+ get_draw_clickable_assets_instructions() : [(DrawAssetInstruction, Clickable)]
+ on_move_action_fired(PlayableCharacter, Int) : Void

2..4

<<abstract>>
PlayableCharacter

- draw_properties: DrawProperties
variant: PlayableCharacterVariant

+ __init__(DrawProperties = None, PlayableCharacterVariant)
+ get_draw_properties(): DrawProperties
+ set_draw_properties(DrawProperties) : Void
+ take_turn([pygame.Rect]) : pygame.Rect
+ get_draw_assets_instructions() : [DrawAssetInstruction]

Dragon

+ take_turn([pygame.Rect]) : pygame.Rect
+ get_draw_assets_instructions() : [DrawAssetInstruction]

Dragon.take_turn: Allows a dragon to take its turn & perform actions. Chit card [for now] hitboxes passed in to ensure a valid move (i.e click) is made. Returns hitbox clicked

DrawProperties

- coordinates: (Int, Int)
- size: (Int, Int)
- rotation: Float

+ __init__((Int, Int), (Int, Int), Float = 0)
+ get_coordinates() : (Int, Int)
+ get_size() : (Int, Int)
+ get_rotation(): Float

NormalTile

+ __init__(Animal, DrawProperties = None)
+ place_character(PlayableCharacter) : Void
+ remove_character() : Void
+ get_draw_assets_instructions() : [DrawAssetInstruction]

GameWorld.run: Contains the maid game loop. Handles policy of player turns, and drawing logic (based on drawable)

<<interface>>
MoveActionHandler

+ on_move_action_fired(PlayableCharacter, Int): Void

<<abstract>>
Tile

- tile_draw_data: DrawProperties
- character: PlayableCharacter
- animal: Animal
- size: (Int, Int)

+ __init__(DrawProperties = None, PlayableCharacter = None, Animal = None)
+ character_on_tile(): PlayableCharacter
+ get_animal_on_tile(): Animal
+ get_draw_data: DrawProperties
+ set_draw_data(DrawProperties) : Void
# get_character_draw_instructions(): [DrawAsseInstruction]
+ place_character(PlayableCharacter) : Void
+ remove_character() : Void
+ get_draw_assets_instructions() : [DrawAsseInstruction]

CaveTile

+ __init__(Animal, DrawProperties = None)
+ place_character(PlayableCharacter) : Void
+ remove_character() : Void
+ get_draw_assets_instructions() : [DrawAssetInstruction]

CaveTile.place_character: Place dragon onto tile and perform check whether it was return back to own cave, trigger win
CaveTile.remove_character: Remove character and configure class to win on arrival of character
CaveTile.get_draw_assets_instructions: Draw the cave tile + animal icon

Tile.__init__: Initialise tile with drawing properties (optional playable character (optional) and animal
Tile.place_character: Place playable character onto tile and perform related functionalities
Tile remove_character: Remove the playable character that's on the tile and perform any related functionalities
Tile.character_on_tile: Returns the playable character on the tile (if any)

MoveActionPublisher

- subscribers: [MoveActionHandler]

+ __init__()
+ subscribe(MoveActionHandler) : Void
+ unsubscribe(MoveActionHandler) : Void
+ notify_subscribers(PlayableCharacter, Int) : Void

MoveActionPublisher.notifySubscribers: Notify subscribers about a move action that needs to be handled. Character and steps to take passed in

EventBus

- win_event_publisher: WinEventPublisher
- move_action_publisher: MoveActionPublisher
- shared: EventBus

+ __init__()
+ publish_win_event(PlayableCharacter) : Void
+ publish_move_action(PlayableCharacter, int): Void
+ add_listener(EventType) : Void
+ shared() : EventBus

EventBus.shared: Get the eventbus singleton
EventBus.publish_win_event: Publish win event onto bus
EventBus.publish_move_action: Publish move action onto bus
EventBus.addListener: Add a listener for a certain event type

WinEventPublisher

- subscribers: [WinEventListener]

+ __init__()
+ subscribe(WinEventListener): Void
+ unsubscribe(WinEventListener) : Void
+ notify_subscribers(PlayableCharacter) : Void

WinEventPublisher.notifySubscribers: Notify subscribers about the player who won

<<interface>>
WinEventListener

+ on_player_win(PlayableCharacter)

WinEventListener.onPlayerWin: On a player win, do something

<<enum>>
EventType

.win
.move_action_fired