Animal .salamander .baby_dragon .spider DrawAssetInstruction AnimalChitCard PirateChitCard asset_path: String - animal: Animal - x: Int + perform_effect(PlayableCharacter) : Void + perform_effect(PlayableCharacter) : Void - size: (Int, Int) - rotate: Float - associated_drawable: DrawableByAsset | None DefaultGameBoard.__init__: First parameter is set main tile sequence. Second parameter is [(starting tile, next tile)]. The second parameter (starting tiles) will be used to make insertions to main tile sequence property. + __init__(String, Int, Int, (Int, Int), Float) DefaultGameBoard.starting_tiles: [starting tiles] + get_asset_path(): String ModularClickableSprite.rect: Dictionary mapping click hit boxes to the object DefaultGameBoard.move_character_by_steps: Moves a dragon from its tile by x number of + get_x_coord: Int associated with it + get_y_coord: Int ChitCard.__init__: Create a chit card with the # number of symbols ModularClickableSprite.on_click: Do something to the object on click DefaultGameBoard.flip_chit_card: Flip a chit card on this game board. Chit card object will be + get_width: Int ChitCard.perform_effect: Perform the chit card's effect on the dragon. ModularClickableSprite.update_click_hitboxes: Update click hitboxes based on obtained by click listener in pygame. + get_height: Int Run by flipChitCard draw instructions (get_draw_instructions) DefaultGameBoard.add_chit_card: Add a chit card to the game board + get_rotation: Float ChiCard.set_flipped: Set flipped + get_associated_drawable: DrawableByAsset ModularClickableSprite DefaultGameBoard <<abstract>> ChitCard main_tile_sequence: [Tile] Entities & Responsibilities + rect: Dictionary<pygame.Rect, starting_tiles: [Tile] DrawableByAsset> - on_game_board: GameBoard GameWorld: Manages game initialisation & admin tasks (e.g game - chit_cards: [ChitCard] = [] - flipped: Bool config, game setup, main game loop, player turn.) - symbol_count: Int + __init__([Tile], [(Tile, Tile)]) + ___init___() <<interface>> + move_character_by_steps(PlayableCharacter, Int) : Void + get_draw_instructions() : [DrawAssetInstruction] Tile: Represents the tiles which the dragons will interact with (stand on) DrawableByAsset.get_draw_instructions: Gets the drawing instructions for DrawableByAsset + get_character_floor_tile(PlayableCharacter) : Tile + on_click() : Void + __init__(GameBoard, Int) drawing an object onto the pygame screen using assets + flip_chit_card(PlayableCharacter, ChitCard) : Void + perform_effect(PlayableCharacter) : Void + update_click_hitboxes() : Void PlayableEntity: Represents the playable entity a player interacts with + get_draw_instructions() : [DrawAssetInstruction] + add_chit_card(ChitCard) + set_flipped(Bool) : Void GameBoard: 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 WinEventXxxx: Publisher and listeners for win event <<abstract>> <<interface>> GameWorld PlayableCharacter Dragon DrawableByAsset: Indicates that the object is drawable by pygame using GameBoard - playable_characters: [PlayableCharacter] - on_game_board: GameBoard + move_character_by_steps(PlayableCharacter, Int) - chit_card: [ChitCard] DrawAssetInstruction: A data class for organising data required for Dragon.take_turn: Allows a dragon to take its turn & perform actions (e.g chit - game_board: GameBoard drawing an asset + __init__(GameBoard) card flipping) + take_turn() : Void + get_character_floor_tile(PlayableCharacter) : Tile + take_turn() : Void + flip_chit_card(PlayableCharacter, ChitCard) : Void ModularClickableSprite: Allows classes to be represented as a sprite that + __init__([PlayableCharacter], [ChitCard], + add_chit_card(ChitCard): Void is clickable on a screen. GameBoard) + get_draw_instructions() : [DrawAssetInstruction] + run(params): Void Patterns Used Observer: WinEventPublisher, WinEventListener • Why?: Don't have to check all starting tiles to see if win occured. GameApplication.__init__: Configure and intialise playable characters, Allows for wins from other sources game board & its tiles & its chit cards. NormalTile GameApplication.run: Contains the main game loop. Handles policy of Singleton: EventBus player turns, and drawing logic (based on drawable) • Why?: Should be one central event bus managing all events <<abstract>> + __init__(PlayableCharacter, Animal) Tile + place_character(PlayableCharacter) : <u>Todo</u> - on_tile: PlayableCharacter - animal: Animal Cardinalities + __init__(PlayableCharacter, Animal) + character_on_tile(): PlayableCharacter CaveTile Upcasts are safe + get_animal_on_tile(): Animal CaveTile.place_character: Place dragon onto tile and perform + remove_character() : Void check whether it was return back to own cave, trigger win + place_character(PlayableCharacter) : + __init__(PlayableCharacter, Animal) + place_character(PlayableCharacter) : Tile. __init__: Initialise tile with playable character (dragon) and animal Tile.place character: Place dragon onto tile and perform related functionalities Tile remove_character: Remove the dragon that's on the tile (if any) Tile.character_on_tile: Returns the dragon on the tile (if any) WinEventPublisher - subscribers: [WinEventListener] <<interface>> WinEventListener + __init__() + subscribe(WinEventListener) + onPlayerWin(PlayableCharacter) + unsubscribe(WinEventListener) WinEventListener.onPlayerWin: On a player win, do + notifySubscribers(PlayableCharacter) something WinEventPublisher.notifySubscribers: Notify subscribers about the player who won EventBus - win_event_publisher: WinEventPublisher EventBus.shared: Get the eventbus singleton - shared: EventBus EventBus.publishWinEvent: Publish win event onto bus EventBus.addListener: Add a listener for a certain event ___init___() + publishWinEvent(PlayableCharacter) : Void + addListener(EventType) : Void + shared() : EventBus <<enum>> EventType

<<enum>>