Updated Sprint 2's Class Diagram for Sprint 3
Legend:
NEW class for Sprint 3 >>
Class
New or Modified Relationship / Cardinalities
association
composition
dependency
Existing Class Updated with new methods/attributes
Class
+ example_attr
- example_method(): None
Notes (will get removed in final ver)
Notes/brief explanation about the modified/new class
Extensions planned to be implemented in Sprint 3: - Timer for both players - Human Value Extension : Tutorial - New God Card: Triton
It's colored so we can differentiate the changes made from sprint 2
Since there are extra win conditions introduced (TIMER WIN), WinConditionStrategy (abstract class) was created
WinConditionStrategy
+ check_win(player: Player, worker: Worker, board: Board): bool
+ check_lose(player: Player, board: Board): bool
TimerWinCondition
+ check_win(player: Player, worker: Worker, board: Board): bool
+ check_lose(player: Player, board: Board): bool
StandardWinCondition
+ check_win(player: Player, worker: Worker, board: Board): bool
+ check_lose(player: Player, board: Board): bool
CompositeWinCondition
- win_conditions: List[WinConditionStrategy]
+ add_win_condition(win_condition: WinConditionStrategy): None
+ remove_win_condition(win_condition: WinConditionStrategy): None
+ check_win(player: Player, worker: Worker, board: Board): bool
+ check_lose(player: Player, board: Board): bool
Composite Win Condition that allows combination of multiple win strategy.
When time runs out = lose
Standard Win Condition (lv2 -> lv3), lose when both workers unable to move
delegates to
WinConditionChecker
- strategy: WinConditionStrategy
+ check_win(player: Player, worker: Worker, board: Board)
+ check_lose(player: Player, board: Board)
+ determine_winner(players: List[Player], current_player: Player, worker: Worker, board: Board): Player
To manage the phases in the game
GamePhaseManager
- actions: Optional[Sequence]
- worker: Optional[Worker]
+ action_sequence(): Optional[Sequence]
+ initialize_turn(action_sequence: Optional[Sequence]): None
+ get_current_phase(): str
+ is_current_phase_optional(): bool
+ advance_phase(): None
+ is_turn_complete(): bool
+ get_current_action(): None
+ has_worker_selected(): bool
+ handle_action_result(result: ActionResult): None
tracks
manages
owns
GameInputHandler
- turn_manager: TurnManager
+ handle_tile_click(tile: Tile): None
- handle_worker_selection(tile: Tile): None
- handle_action_execution(tile: Tile): None
To handle user input such as worker selection, tile clicking, action execution
TutorialManager
- steps: List[TutorialStep]
- current_step_index: int
- observers: List[TutorialObserver]
- is_tutorial_complete: bool
- introduction_clicked: bool
- tutorial_type: str
+ add_observer(observer: TutorialObserver): None
+ remove_observer(observer: TutorialObserver): None
+ get_current_step(): TutorialStep
+ get_current_instructions(): str
- advance_step(): None
+ complete_tutorial(): None
- notify_step_changed(step: TutorialStep): None
- notify_tutorial_completed(): None
notifies
contains
TutorialStep abstract class
TutorialStep
+ get_step_name(): str
+ get_instructions(): str
+ is_valid_tile_click(tile: Tile, current_player: Player, current_worker: Worker, phase: str): bool
+ get_highlighted_tiles(current_player: Player, current_worker: Worker, phase: str, board: Board): bool
+ is_step_complete(current_player: Player, current_worker: Worker, phase: str, board: Board): bool
+ handles_click_progression(): bool
+ handle_tile_click(tile: Tile, current_player: Player, current_worker: Worker, phase: str): None
+ should_auto_advance(): bool
TutorialObserver
+ on_tutorial_step_changed(step: TutorialStep): None
+ on_tutorial_completed(): None
TutorialUIAdapter
- app: SantoriniApp
+ on_tutorial_step_changed(step: TutorialStep): None
+ on_tutorial_completed(): None
SelectWorkerStep
MoveWorkerStep
MoveToTrapStep
- move_completed: bool
- demonstration_shown: bool
MoveToLevel3Step
IntroductionStep
- tutorial_type: str
BuildStep
CompletionStep
- tutorial_type: str
+ get_step_name(): str
+ get_instructions(): str
+ is_valid_tile_click(tile: Tile, current_player: Player, current_worker: Worker, phase: str): bool
+ get_highlighted_tiles(current_player: Player, current_worker: Worker, phase: str, board: Board): bool
+ is_step_complete(current_player: Player, current_worker: Worker, phase: str, board: Board): bool
+ handle_tile_click(tile: Tile, current_player: Player, current_worker: Worker, phase: str): None
+ handles_click_progression(): bool
+ should_auto_advance(): bool
Specific steps for tutorial game mode
Game mode (tutorial vs standard)
GameMode
+ initialize_game(players: List[Player], board: Board): None
+ is_valid_tile_click(tile: Tile, current_player: Player, current_worker: Worker, phase: str): bool
+ get_highlighted_tiles(current_player: Player, current_worker: Worker, phase: str, board: Board): bool
+ get_guidance_message(tile: Tile, current_player: Player, current_worker: Worker, phase: str): str
+ should_end_game(): bool
+ get_mode_name(): str
+ handle_post_action_update(urrent_player: Player, current_worker: Worker, phase: str, board: Board): None
+ handle_tile_click(current_player: Player, current_worker: Worker, phase: str, board: Board): None
StandardGameMode
+ initialize_game(players: List[Player], board: Board): None
+ get_guidance_message(tile: Tile, current_player: Player, current_worker: Worker, phase: str): str
+ should_end_game(): bool
+ get_mode_name(): str
- place_random_workers(player: Player, board: Board): None
TutorialGameMode
- tutorial_manager: TutorialManager
- tutorial_type: str
+ initialize_game(players: List[Player], board: Board): None
+ get_guidance_message(tile: Tile, current_player: Player, current_worker: Worker, phase: str): str
+ should_end_game(): bool
+ get_mode_name(): str
- setup_tutorial_board(player: Player, board: Board): None
- clear_board(board: Board): None
+ handle_post_action_update(urrent_player: Player, current_worker: Worker, phase: str, board: Board): None
+ handle_tile_click(current_player: Player, current_worker: Worker, phase: str, board: Board): None
Board
- tiles: dict[tuple[int, int], Tile]
- width: int
- height: int
+ in_bounds(pos: Position): bool
+ get_all_empty_tiles(): List[Tile]
+ get_tile(pos: Position): Tile | None
consists of
creates
creates/register
has
contains
Validator
+ get_valid_move_tiles(worker: Worker, board: Board): list[Tiles]
+ get_valid_build_tiles(worker: Worker, board: Board): list[Tiles]
Sequence can now handle action result
Sequence
- items: List[Any]
- index: int
+ current(): Any
+ advance(): None
+ reset_index(): None
+ handle_action_result(result: ActionResult): None
+ is_complete(): bool
players
actions
Modified TurnManager to specifically only handle turns
TurnManager
- players: Sequence
- board: Board
- phase_manager: GamePhaseManager
- win_checker: WinConditionChecker
- timer_manager: TimerManager
+ get_phase(): str
+ current_phase_optional(): bool
+ skip_phase(): None
+ start_turn(): None
+ end_turn(): None
+ get_game_result(): Player
Game
- board: Board
- players: list[Player]
- turn_manager: TurnManager
- god_card_deck: GodCardDeck
- timer_manager: TimerManager
- input_handler: GameInputHandler
+ get_current_phase(): str
+ current_phase_optional(): bool
+ skip_phase(): bool
- place_random_workers(player: Player)
- pick_random_god(player: Player)
+ click_cell(bx: int, by: int): bool
+ selected_worker_pos(): Tuple(int,int) | None
initializes and contains
SantoriniApp
- root: tk.Tk
- game: Game
- rect_ids: dict[tuple[int, int], int]
- player1_name_var: tk.StringVar
- player2_name_var: tk.StringVar
- player1_color_var: tk.StringVar
- player2_color_var: tk.StringVar
- main_menu_frame: tk.Frame
- setup_frame: tk.Frame
- game_frame: tk.Frame
- canvas: tk.Canvas
- skip_button: tk.Button
- turn_label: tk.Label
- phase_label: tk.Label
- god_name: tk.Label
- worker_icon: tk.PhotoImage
+ start_game(): None
- build_main_menu_ui(): None
- build_setup_ui(): None
- show_setup_screen(): None
- build_game_ui(): None
- draw_board(): None
+ on_click(evt: tk.Event): None
+ on_skip(): None
Plan to change the app class (more OOP based)
ActionResult
- additional_actions
+ has_additional_actions(): List[Action
+ is_empty(): bool
To handle post action execution
Created a god card factory to handle addition of new cards
GodCardFactory
- registered_cards
+ create_card()
+ register_card()
+ get_available_card_names()
Uses GodCardFactory to get the GodCards available
GodCardDeck
- cards: List[GodCard]
+ draw(): GodCard
+ add_card(god_card: GodCard): None
+ is_empty(): None
+ remaining(): List[GodCard]
Action
- optional: bool
+ execute(worker: Worker, board: Board, target_tile: Tile): ActionResult
+ validate(worker: Worker, board: Board, target_tile: Tile): None
+ get_name(): str
GodCard
- name: str
- description
+ get_action_sequence(): List[Action]
wields
MoveAction
+ execute(worker: Worker, board: Board, target_tile: Tile): ActionResult
+ validate(worker: Worker, board: Board, target_tile: Tile): bool
Modified Action classes execute() method to return ActionResult
BuildAction
+ execute(worker: Worker, board: Board, target_tile: Tile): ActionResult
+ validate(worker: Worker, board: Board, target_tile: Tile): bool
TritonMoveAction
+ execute(worker: Worker, board: Board, tile: Tile)
- is_perimeter_space(position: Position, board; Board)
Since Triton's Move Action has different validations vs Artemis, added a new MoveAction specifically for Triton
ArtemisMoveAction
+ validate(worker: Worker, board: Board, target_tile: Tile): None
DemeterBuildAction
+ validate(worker: Worker, board: Board, target_tile: Tile): None
Artemis
Triton
Demeter
StandardGodCard
Added Triton God class (implements abstract God class)
Added standard god (no powers) for tutorial
Position
- x: int
- y: int
occupies
Tile
- position: Position
- worker: Worker | None
- building: Building | None
+ has_worker(): bool
holds
Building
+ has_dome(): bool
+ get_level(): int
+ increase_level(): None
Block
- level: int
Dome
- level: int
<<enumeration>>
Color
Each player will have their own PlayerTimer
Player
- player_name: str
- player_age: str
- player_color: Color
- workers: List[Worker] | None
- player_god: GodCard | None
- timer: PlayerTimer
+ add_worker(worker: Worker): None
controls
selects
controls timer of
owns
Worker
- position: Position
- previous_position: Position
- previous_build_position: Position
- color: Color
- set_position(new_position: Position): None
TimerInfo and PlayerTimerInfo are TypedDict
TimerManager
- players: List[Player]
- current_timer_player: Player
- timer_enabled: bool
+ start_player_timer(player: Player): None
+ pause_current_timer(): None
+ switch_timer_to_player(player: Player): None
+ get_current_player_timer_info(): TimerInfo
+ get_all_players_timer_info(): Dict[str, PlayerTimerInfo]
+ check_for_timer_expiration(): Player
+ has_any_timer_expired: bool
+ get_expired_players(): List[Player]
- format_time(seconds: float): str
+ reset_all_timers(): None
+ get_players_with_timers(): List[Player]
PlayerTimer
- time_limit: float
- remaining_time: float
- start_time: float
- is_active: bool
- is_expired: bool
- start(): None
- pause(): None
- reset(): None
- add_time(seconds: float): None
- get_formatted_time(): str
Stores the information for the timer of each player