10 Building our own game (1/4) - Game Layout

Functional and Implementation Guidelines

Functional Guidelines

1. Remove all of the test game logic and game objects created so far
   1. Let’s start the implementation of our first game - Chess!
2. Create a set of helper functionalities for the GameBoard (BoardUtils)
   1. get board tile position from absolute window coordinates and vice-versa
   2. is a point inside the game board or not
   3. etc…
3. Implement a basic functionality for a Chess Piece. It should support:
   1. self-draw
   2. working with events
   3. functionality for moving the chess piece within the game board movement tiles
4. Implement a functionality that housekeeps the logic for all chess pieces
   1. object lifetime & creation
   2. object movement
   3. event handling
5. Implement a “target” animation, which marks the current selected chess piece
   1. start/show the animation once a piece is selected
   2. stop/hide the animation once a piece in unselected

Implementation Guidelines

1. Clean up the Game class (keep the folder structure)
2. Examine the new assets - the chessBoard, whitePieces, blackPieces
3. Implement a simple GameBoard class - containing the chessBoard image

| class GameBoard {  Image \_boardImg;  }; |
| --- |

1. BoardPos struct (similar for Point struct)

| struct BoardPos {  int32\_t row;  int32\_t col;  }; |
| --- |

1. BoardUtils set of functions

| static BoardPos getBoardPos(const Point &absPos);  static Point getAbsPos(const BoardPos& boardPos);  static bool isInsideBoard(const BoardPos& boardPos);  static bool isInsideBoard(const Point& absPos); |
| --- |

* 1. Common Defines for BoardUtils

| constexpr auto BOARD\_SIZE = 8;  constexpr auto FIRST\_TILE\_X\_POS = 58;  constexpr auto FIRST\_TILE\_Y\_POS = 60;  constexpr auto TILE\_SIZE = 98; |
| --- |

1. ChessDefines

| enum PlayerId {  WHITE\_PLAYER\_ID = 0, BLACK\_PLAYER\_ID,  PLAYERS\_COUNT  };  enum class PieceType : uint8\_t {  KING, QUEEN, BISHOP, KNIGHT, ROOK, PAWN  }; |
| --- |

1. ChessPiece class
   1. Structure

| class ChessPiece {  Image \_pieceImg;  BoardPos \_boardPos;  int32\_t \_playerId;  PieceType \_pieceType;  }; |
| --- |

* 1. Public API

| void draw();  bool containsEvent(const InputEvent &e) const;  void setBoardPos(const BoardPos &boardPos);  //+ getters |
| --- |

PieceHandler class - responsible for handling logic about individual pieces

* 1. Defines

| enum PieceHandlerDefines {  constexpr auto STARTING\_PIECES\_COUNT = 16;  constexpr auto PAWNS\_COUNT = 8;  constexpr auto WHITE\_PLAYER\_START\_PAWN\_ROW = 6;  constexpr auto BLACK\_PLAYER\_START\_PAWN\_ROW = 1;  }; |
| --- |

* 1. Structure

| class PieceHandler {  using PlayerPieces = std::vector<ChessPiece>;  std::array<PlayerPieces, Defines::PLAYERS\_COUNT> \_pieces;  int32\_t \_selectedPieceId = 0;  int32\_t \_selectedPiecePlayerId = 0;  bool \_isPieceGrabbed = false;  }; |
| --- |

1. Implement a target image in the GameBoard struct

| class GameBoard {  Image \_boardImg;  Image \_targetImg;  }; |
| --- |

1. Activate the target when a piece is grabbed/ungrabbed
2. GameBoardInterface

| virtual void onPieceGrabbed(const BoardPos &boardPos) = 0;  virtual void onPieceUngrabbed() = 0; |
| --- |

1. The PieceHandler should invoke the GameBoardProxy interface

| class PieceHandler {  std::array<PlayerPieces, Defines::PLAYERS\_COUNT> \_pieces;  GameBoardInterface \*\_gameBoardInterface = nullptr;  int32\_t \_selectedPieceId = 0;  int32\_t \_selectedPiecePlayerId = 0;  bool \_isPieceGrabbed = false;  }; |
| --- |

1. Bonus: make the target blink when active