**Felipe Lima – CSCI 3010 HW#1 part 1**

**Player Class**

* **Player (const std::*string* *name*, const bool *is\_human*);**
  + Initializes a player with the passed name and whether it is a human or not.
  + Initiate the player with name, points, is-human, and position
* **std::*string* get\_name() const {return name\_; }**
  + Returns the name of the player as a string
* **int get\_points() const {return points\_; }**
  + Returns the points of the player as an int
* ***Position* get\_position() const {return pos\_; }**
  + Returns the position of the player as an instance of the struct Position
* **bool is\_human() const {return is\_human\_; }**
  + Returns true or false for whether the player is human
* **void ChangePoints(const int x);** 
  + Updates the points of the player by updating “points\_” with the value passed in the function
* **void SetPosition(Position pos);**
  + Sets a new position for the player by updating the value of “pos\_” with the value passed in the function.
* **std::string ToRelativePosition(Position other);**
  + Translates “other” into a direction relative to the player.
  + Compares current position of the player with position passed into the function and returns a direction (i.e. “Up”, “down” etc.)
  + (i.e. if current = x,y and other = x+1, y -> return “Down”)
* **std::string Stringify();**
  + Converts this instance of “Player” into a string representing its name and points
  + Returns a string saying “Player had x points”

**std::*string* SquareTypeStringify(*SquareType* *sq*);**

Return a string representation of a given SquareType

Compares the SquareType passed into the function and returns the corresponding representation in string or emoji

(i.e. if SquareType == Wall return “wall” or corresponding emoji)

**Board Class**

* **Board();**
  + Constructor. Initializes a board.
  + Initializes the board by populating the squares with 10% chance of being a treasure, 20% chance of being a wall. (use rand() to dp this)
  + Randomly inserts the desired number of enemies into the board
  + Insert the human at 0,0 and the exit at 3,3
* **int get\_rows() const {return 4; }** 
  + returns the number of rows on the board
* **int get\_cols() const {return 4; }** 
  + returns the number of columns on the board
* ***SquareType* get\_square\_value(*Position* *pos*) const;** 
  + Returns the type of the square of the position passed into the function
* **void SetSquareValue(*Position* *pos*, *SquareType* *value*);**
  + sets the type of the square passed as the position parameter as the squaretype passed to the function
  + change the current square type
* **std::vector<*Position*> GetMoves(*Player* \**p*);**
  + gets the possible moves for the player
  + Checks for every possibility and
    - Calls get\_square\_value
    - If its not a wall or outside of the board, return as a possible entry
* **bool MovePlayer(*Player* \**p*, *Position* *pos*);** 
  + moves the player on the board to the desired position and return true if successful, false otherwise.
    - Calls get\_square\_value
  + If the move is possible, change the position of the player to the updated position
  + If not return false
* ***SquareType* GetExitOccupant();**
  + Gets the square type of the exit square
    - Doesn’t have to but can call get\_square\_value with the exit position
* **friend std::*ostream*& operator<<(std::*ostream*& *os*, const *Board* &*b*);**
  + Overloads the operator << so we can print the element of the Board class
    - Calls std::*string* SquareTypeStringify(*SquareType* *sq*);
  + Print out the board (like a 2D array) with the emoji

**Maze Class**

* **Maze();**
  + Constructor. Initializes a maze.
  + Initializing a maze initializes a new game every time.
  + This means creating players resetting points and names
  + Might call Board() and initialize the board from within Maze()
* **void NewGame(Player \*human, const int enemies);**
  + Initializes a new game with the human player and the number of enemies
  + Creates the player and sets the number of enemies
* **void TakeTurn(Player \*p);**
  + Have the player passed into the function to take their turn
  + Call MovePlayer(), GetMoves() and the consequent functions called by these two
  + Ask a player a direction and execute the move
* **Player \* GetNextPlayer();**
  + Get the next player in the right order
  + Set the current player as the next in line (human, e1, e2)
* **bool IsGameOver();**
  + return true if the game is over, that it, if a human reached the exit or the enemies eliminated the humans.
    - Calls GetExitOccupant();
    - If the occupant is human, end game
    - Check if human and enemy occupy same space, if so, end game
* **std::string GenerateReport();**
  + Reports the points for every player.
    - Call Stringify() for every player
* **friend std::ostream& operator<<(std::ostream& os, const Maze &m);**
  + Overload the operator << to print elements from Maze class.