Nicholas Jenkins
CS 250
Grid Racers: Top-Down Design
December 6, 2013

Changes to the Top-Down Design

1. Main
    a. The main program was reduced to initializing the game and receiving input from the command line prompts
        i. The rule printing was removed as it was unnecessary
        ii. All printing commands were moved elsewhere within the program
        iii. All game-play commands were removed
2. Game
    a. The move determining routine was moved to the Player section
    b. The game-play order was focused in this portion of the program
    c. All coding that dealt with finding the next player and calling for the player's move was moved to this section
3. Track
    a. Track wall values set to X instead of having a value of 1,000,000
4. Player
    a. The move vector routine was moved to within the Player section
5. Human Player
    a. No major changes
6. Computer Player
    a. No major changes

Main

```
int main (int argc, char** argv)
//Initialize grid race game
//*determine the number of argument sent to command line
//**send input and output streams to Game constructor
//**start game
```

Game

```
Game::Game()
//*Constructor for game data type
//**assign input and output streams to appropriate constructor calls
//**Initializes the game track
//**Initializes all three players (computer 1, computer 3, and human 2)

void Game::playGame()
//*Administers game playing routin
//**Determines the next player to make a move
//**Calls the track printer to show the track for player(s)
```

//**Determines if any of the players have completed the race
//***Game stops if a player has finished

bool Game::gameEnded()
//*Checks the position of each player to see if they have reached the finish line
//*Returns a true if a player has finished the race

void Game::assignStart()
//*Finds the starting position for each player
//**Searchs the initialized game board to find player number values
//**Sets the position of each player to location of the numbers as they are found

Track
Track::Track()
//*Constructs the Track data type
//**Reads the track from the input stream to gauge the dimensions of the track
//***The dimensions of the track are stored for the construction of the Track
//**The Track is constructed from a two dimensional dynamic memory array of Cells
//***The cells are populated as the input stream is re-read to create the Track
//****The player numbers are inserted
//****The walls are inserted
//****The finish line is inserted
//*****The default distance from the finish line is set to -1
//*****The finish line has a value of zero from finish

Track::~Track()
//*Destructs the two dimensional dynamic array created by the constructor
//**The arrays are deleted from inside the arrays first and then the outer arrays are deleted

int Track::getWidth()
//*Returns the width of the track

void Track::setWidth()
//*Sets the width of the track

int Track::getHeight()
//*Returns the height of the track

void Track::setHeight()

//*Sets the height of the track

void Track::printTrack()
//*Prints the contents of the track cells
//**Prints the walls and the players starting from the top left corner

void Track::assessTrack()
//*Determines the distance from the finish for every cell in the track
//**Finds the finish line
//**Determines all of the adjacent cells to the finish line that are not walls
//**Sets the value of the cells adjacent to the finish line to one
//**Searches for the cells that have the last value assigned
//***Finds all of the adjacent cells and assigns a value one higher than the last
//**Continues working until all cells are populated

Player
Player::Player()
//*Constructs the Player data type
//**Sets all values to their default
//**Position & Velocity set to zero
//**Maximum speed set to five

char Player::getNumber()
//*Returns the car number of the player

void Player::setNumber()
//*Sets the car number of the player

int Player::getMaxSpeed()
//*Returns the current maximum speed of the player

void Player::reduceMaxSpeed()
//*Reduces the maximum speed of the player
//**Max speed cannot fall below 1

Position Player::getPosition()
//*Returns the current position of the player

void Player::setPosition ()

//*Sets the current position of the player
//**Moves the players number on the Track

Velocity Player::getVelocity()
//*Returns the current velocity of the player

void Player::setVelocity ()
//*Sets the velocity of the player

void Player::print()
//*Prints out the needed information about the player
//**Number, Position, Velocity

void Player::makeMove()
//*Carries out a move called by the Player

int Player::codeFinder()
//*Returns a coded value for each move type
//**0 if the there are no obstructions in the player's path
//**1 if there is a wall in the player's path
//**2 if there is another player in the player's path
//**3 if the finish line is in player's path

void Player::goToFinishLine()
//*Moves the player immediately to the finish line if the finish line is crossed during their turn

void Player::setFinishLine()
//*Assigns the finish line position for the player based on their move

Position Player::getFinishLine()
//*Returns the position of the finish line

int Player::isLegal()
//*Takes a move proposed by the player and determines its legality
//**The Cells are stepped through based on the move sent by the player
//**The Code is determined via function call based on the player's move

Human Player
humanPlayer::humanPlayer()

//*Constructs the humanPlayer data type
//**Sets all values to their default
//**Calls for a Player to be constructed

void humanPlayer::getNextMove()
//*Requests input from the user
//**Checks to see if the input is valid
//**Attempts to make the move if the input values are valid

void humanPlayer::attemptMove(Velocity v, Track& t, ostream& out){
//*Attempts to move the player based upon their input
//*Enforces any rule violations if committed by the player

Computer Player
computerPlayer::computerPlayer()
//*Constructs the computerPlayer data type
//**Sets all values to their default
//**Calls for a Player to be constructed

void computerPlayer::getNextMove()
//*Determines the move to be made by the computer
//**Checks all five move possibilities to see if they are legal
//**Compiles the list of legal moves
//**The move is chosen based up the characteristics of the Player 1 vs Player 3

void computerPlayer::attemptMove()
//*Attempts to move the player based upon the move chosen by the computer
//*Enforces any rule violations if committed by the player

void computerPlayer::moveSort()
//*Sorts the list of compiled legal moves determined by the computer player
//**Sorts based up the 'driving technique' of each player
//**Sorts the moves by distance from the finish so the moves are progressive
//***Player 1 drives with maximum speed
//***Player 3 drives with maximum maneuverability