# Game Design Phase from GroupName = “GrOuPnAmE”

## 20+ Must Have Features:

These are the bare minimum (must have!) features for the game to run

1. Player must be able to move up, right, down and left
2. Player cannot move off the edge of the grid
3. Player starts the level on top of a floor or goal tile
4. Player cannot move onto a brick
5. Player cannot push a brick or goal
6. Player can push a box
7. A box cannot be pushed into a wall
8. A box cannot be pushed into another box
9. A box cannot be pushed off the edge of the grid
10. Brick, Goal, Box, Player and Floor Elements must start set where the level file wants them to be
11. The game must start with at least one goal
12. The game must start with at least one box
13. The game must start with an equal number of goals and boxes
14. The game must start with one player
15. The level is complete when all the boxes are on all the goals
16. You can start a game
17. You can finish a game
18. You can use the arrow keys to move the player
19. The game will record a list of your moves
20. You can undo one or more moves
21. The game can load a file from the file handler API

## Test Case Checklist:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Player | Can move | Up |
| 2 | Player | Can move | Right |
| 3 | Player | Can move | Down |
| 4 | Player | Can move | Left |
| 5 | Player | Cannot move off the | Top edge of the grid |
| 6 | Player | Cannot move off the | Right edge of the grid |
| 7 | Player | Cannot move off the | Bottom edge of the grid |
| 8 | Player | Cannot move off the | Left edge of the grid |
| 9 | Player | Starts the level on top of | Floor or Goal tile |
| 10 | Player | Cannot move Up if there’s a brick | Above the player |
| 11 | Player | Cannot move Right if there’s a brick | Right to the player |
| 12 | Player | Cannot move Down if there’s a brick | Below the player |
| 13 | Player | Cannot move Left if there’s a brick | Left to the player |
| 14 | Player | Can push a box | Up |
| 15 | Player | Can push a box | Right |
| 16 | Player | Can push a box | Down |
| 17 | Player | Can push a box | Left |
| 18 | Player | Cannot push a | Brick |
| 19 | Player | Cannot push a | Goal |
| 20 | Box | Cannot be pushed into a | Wall |
| 21 | Box | Cannot be pushed into a | Box |
| 22 | Box | Cannot move off the | Top edge of the grid |
| 23 | Box | Cannot move off the | Right edge of the grid |
| 24 | Box | Cannot move off the | Bottom edge of the grid |
| 25 | Box | Cannot move off the | Left edge of the grid |
| 26 | Brick # | Element in the game should be where | The LevelDesign string stated it was to be |
| 27 | Goal . | Element in the game should be where | The LevelDesign string stated it was to be |
| 28 | Box $ | Element in the game should be where | The LevelDesign string stated it was to be |
| 29 | Floor \_ | Element in the game should be where | The LevelDesign string stated it was to be |
| 30 | Level | Has at least one | Goal |
| 31 | Level | Has at least one | Box |
| 32 | Level | Has equal number of | Goal and Box |
| 33 | Level | Has one | Player |
| 34 | Level | Is complete when | All boxes are on all goals |
| 35 | You | Can | Start a game |
| 36 | You | Can | Finish a game |
| 37 | You | Can | Undo a move |
| 38 | You | Can | Undo more than 1 move |
| 39 | You | Cannot | Undo more moves than you’ve done |
| 40 | LevelKeystrokes | CheckSaved |  |
| 41 | FileAPI | Does | Exist |
| 42 | FileAPI | Has | GameFiles |
| 43 | FileAPI | Has | Methods |

## Test Case Code:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using SokobanGame;

namespace SokobanTests

{

/\*

\* NOTES:

\*

\* To create a new level it requires certain attributes:

\* rowWidth is how WIDE the level is, e.g. how many columns if it was a table

\* rowCount is how TALL the level is, e.g. how many rows if it was a table

\* playerPosX and playerPosY is what cell the player starts in, e.g. 0, 0 would be top left

\*

\* levelDesign is a string:

\*

\* #####

\* # #

\* # #

\* # #

\* ##### for example becomes "###### ## ## ######" when flattened into one string

\* and it's 5x5 in size, so the rowWidth would be 5, and the rowCount would be 5

\* then if you want the player to start in the middle, playerPosX and playerPosY would both be 2

\*

\* To move the player, call the MovePlayer function

\* Pass the function a number, which represents the DIRECTION to move in

\* 1 = Up, 2 = Right, 3 = Down, 4 = Left

\* At some point this will surely be replaced by something like Keys.Up etc.

\*

\*/

[TestClass]

public class SokobanTests

{

// Player Moves Up Unit Test

[TestMethod]

public void MovePlayer\_Up\_UpdatesPlayerPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 2;

int playerPosY = 2;

string levelDesign = "###### ## ## ######";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(1); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosY();

Assert.AreEqual(expected, actual, "Player failed to move up correctly");

}

// Player Moves Right Unit Test

[TestMethod]

public void MovePlayer\_Right\_UpdatesPlayerPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 2;

int playerPosY = 2;

string levelDesign = "###### ## ## ######";

int expected = 3;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosX();

Assert.AreEqual(expected, actual, "Player failed to move right correctly");

}

// Player Moves Down Unit Test

[TestMethod]

public void MovePlayer\_Down\_UpdatesPlayerPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 2;

int playerPosY = 2;

string levelDesign = "###### ## ## ######";

int expected = 3;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(3); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosY();

Assert.AreEqual(expected, actual, "Player failed to move down correctly");

}

// Player Moves Left Unit Test

[TestMethod]

public void MovePlayer\_Left\_UpdatesPlayerPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 2;

int playerPosY = 2;

string levelDesign = "###### ## ## ######";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(4); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosX();

Assert.AreEqual(expected, actual, "Player failed to move left correctly");

}

// Wall Above Player, and Player Tries to Move Up

[TestMethod]

public void MovePlayer\_Up\_IntoWall\_UpdatesPlayerPosition()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "#### ####";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(1); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosY();

Assert.AreEqual(expected, actual, "Wall above player failed to block player");

}

// Wall to Right of Player, and Player Tries to Move Right

[TestMethod]

public void MovePlayer\_Right\_IntoWall\_UpdatesPlayerPosition()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "#### ####";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosX();

Assert.AreEqual(expected, actual, "Wall to right of player failed to block player");

}

// Wall Under Player, and Player Tries to Move Down

[TestMethod]

public void MovePlayer\_Down\_IntoWall\_UpdatesPlayerPosition()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "#### ####";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(3); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosY();

Assert.AreEqual(expected, actual, "Wall below player failed to block player");

}

// Wall to Left of Player, and Player Tries to Move Left

[TestMethod]

public void MovePlayer\_Left\_IntoWall\_UpdatesPlayerPosition()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "#### ####";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(4); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetPlayerPosX();

Assert.AreEqual(expected, actual, "Wall to left of player failed to block player");

}

// Brick Element in Game Where LevelDesign String Wants it to be

[TestMethod]

public void Brick\_LoadedIntoGame\_InCorrectLocation()

{

int rowWidth = 2;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " #";

char expected = '#';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

char actual = level.WhatIsHere(1, 0);

Assert.AreEqual(expected, actual, "Brick not in expected location");

}

// Box Element in Game Where LevelDesign String Wants it to be

[TestMethod]

public void Box\_LoadedIntoGame\_InCorrectLocation()

{

int rowWidth = 2;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " $";

char expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

char actual = level.WhatIsHere(1, 0);

Assert.AreEqual(expected, actual, "Box not in expected location");

}

// Goal Element in Game Where LevelDesign String Wants it to be

[TestMethod]

public void Goal\_LoadedIntoGame\_InCorrectLocation()

{

int rowWidth = 2;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " .";

char expected = '.';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

char actual = level.WhatIsHere(1, 0);

Assert.AreEqual(expected, actual, "Goal not in expected location");

}

// Floor Element in Game Where LevelDesign String Wants it to be

[TestMethod]

public void Player\_LoadedIntoGame\_InCorrectLocation()

{

int rowWidth = 2;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " \_";

char expected = '\_';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

char actual = level.WhatIsHere(1, 0);

Assert.AreEqual(expected, actual, "Floor not in expected location");

}

// Same amount of boxes as goals

[TestMethod]

public void CheckAmountofBoxesAndGoals()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " $.";

bool expected = true;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

bool actual = (level.AllMyBoxes.Count == level.AllMyGoals.Count);

Assert.AreEqual(expected, actual, "The box to goal ratio does not match");

}

//Tests that all goals are completed

[TestMethod]

public void CheckCompletedGoals()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " $.";

bool expected = true;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2);

bool actual = level.IsComplete();

Assert.AreEqual(expected, actual, "The level did not complete when box went onto goal");

}

//Tests that there is a box element in the level

[TestMethod]

public void IsBoxElementPresent()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " $";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

int actual = level.AllMyBoxes.Count;

Assert.AreEqual(expected, actual, "A box element is not present");

}

//Tests that there is a Goal element in the level

[TestMethod]

public void IsGoalElementPresent()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " .";

int expected = '.';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

int actual = level.WhatIsHere(1, 0);

Assert.AreEqual(expected, actual, "A goal element is not present");

}

//checks that the player cannot exit out the top of the level

[TestMethod]

public void CheckPlayerCantExitTop()

{

int rowWidth = 1;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " ";

int expected = 0;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(1);

int actual = level.PlayerPosY;

Assert.AreEqual(expected, actual, "The Player moved out the top of the level");

}

//checks that the player cannot exit out the left of the level

[TestMethod]

public void CheckPlayerCantExitRight()

{

int rowWidth = 1;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " ";

int expected = 0;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2);

int actual = level.PlayerPosX;

Assert.AreEqual(expected, actual, "The Player moved out the right of the level");

}

//checks that the player cannot exit out the Right of the level

[TestMethod]

public void CheckPlayerCantExitBottom()

{

int rowWidth = 1;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " ";

int expected = 0;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(3);

int actual = level.PlayerPosY;

Assert.AreEqual(expected, actual, "The Player moved out the bottom of the level");

}

//checks that the player cannot exit out the Right of the level

[TestMethod]

public void CheckPlayerCantExitLeft()

{

int rowWidth = 1;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " ";

int expected = 0;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(1);

int actual = level.PlayerPosX;

Assert.AreEqual(expected, actual, "The Player moved out the left of the level");

}

//checks that the player cannot push a box out the top of the level

[TestMethod]

public void Player\_PushingBox\_Up\_WontPushBoxOutOfBounds()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "$$$$ $$$$";

char expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(1);

int actual = level.WhatIsHere(1, 0);

Assert.AreEqual(expected, actual, "The Player pushed the box past top of bounds");

}

//checks that the player cannot push a box out the top of the level

[TestMethod]

public void Player\_PushingBox\_Right\_WontPushBoxOutOfBounds()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "$$$$ $$$$";

char expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2);

int actual = level.WhatIsHere(2, 1);

Assert.AreEqual(expected, actual, "The Player pushed the box past right of bounds");

}

//checks that the player cannot push a box out the top of the level

[TestMethod]

public void Player\_PushingBox\_Down\_WontPushBoxOutOfBounds()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "$$$$ $$$$";

char expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(3);

int actual = level.WhatIsHere(1, 2);

Assert.AreEqual(expected, actual, "The Player pushed the box past bottom of bounds");

}

//checks that the player cannot push a box out the top of the level

[TestMethod]

public void Player\_PushingBox\_Left\_WontPushBoxOutOfBounds()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "$$$$ $$$$";

char expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(4);

int actual = level.WhatIsHere(0, 1);

Assert.AreEqual(expected, actual, "The Player pushed the box past left of bounds");

}

// Player pushes a box Up Unit Test

[TestMethod]

public void MovePlayer\_Up\_UpdatesBoxPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 2;

int playerPosY = 3;

string levelDesign = "###### ## $ ## ######";

int expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(1); // 1 is up, 2 is right, 3 is down, 4 is left

//check box moved up

int actual = level.WhatIsHere(2, 1);

Assert.AreEqual(expected, actual, "Box failed to move up correctly");

}

// Player pushes a box Down Unit Test

[TestMethod]

public void MovePlayer\_Down\_UpdatesBoxPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 2;

int playerPosY = 1;

string levelDesign = "###### ## $ ## ######";

int expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(3); // 1 is up, 2 is right, 3 is down, 4 is left

//check box moved down

int actual = level.WhatIsHere(2, 3);

Assert.AreEqual(expected, actual, "Box failed to move down correctly");

}

// Player pushes a box Right Unit Test

[TestMethod]

public void MovePlayer\_Right\_UpdatesBoxPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 1;

int playerPosY = 2;

string levelDesign = "###### ## $ ## ######";

int expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

//check box moved right

int actual = level.WhatIsHere(3, 2);

Assert.AreEqual(expected, actual, "Box failed to move Right correctly");

}

// Player pushes a box Left Unit Test

[TestMethod]

public void MovePlayer\_Left\_UpdatesBoxPosition()

{

int rowWidth = 5;

int rowCount = 5;

int playerPosX = 3;

int playerPosY = 2;

string levelDesign = "###### ## $ ## ######";

int expected = '$';

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(4); // 1 is up, 2 is right, 3 is down, 4 is left

//check box moved left

int actual = level.WhatIsHere(1, 2);

Assert.AreEqual(expected, actual, "Box failed to move Left correctly");

}

// Level has one Player position

[TestMethod]

public void CheckPlayerValue()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 2;

int playerPosY = 0;

string levelDesign = "# \_";

bool expected = true;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

//bool actual = (level.PlayerPosX != null && level.PlayerPosY != null);

bool actual = (level.PlayerPosX != 0 && level.PlayerPosY != 0);

Assert.AreEqual(expected, actual, "The Player does not have a position on the board");

}

// Player Starts the level on top of Floor or Goal Tile \_ or .

[TestMethod]

public void CheckPlayerStartingOnTop()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 2;

int playerPosY = 0;

string levelDesign = "# \_";

bool expected = true;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

bool actual = (level.WhatIsHere(playerPosX, playerPosY) == '\_' || level.WhatIsHere(playerPosX, playerPosY) == '.');

Assert.AreEqual(expected, actual, "The Player does not start on a Floor or Goal tile");

}

// Box Cannot be pushed into a Wall

[TestMethod]

public void CheckBoxNotPushedIntoWall()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = "\_$#";

bool expected = true;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2);

//Player should not move so should have same starting position

//Only need to check the x position because player moving right

bool actual = (level.PlayerPosX == playerPosX);

Assert.AreEqual(expected, actual, "The Box is being pushed into a wall");

}

// Box Cannot be pushed into a Box

[TestMethod]

public void CheckBoxNotPushedIntoBox()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = "\_$$";

bool expected = true;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2);

//Player should not move so should have same starting position

//Only need to check the x position because player moving right

bool actual = (level.PlayerPosX == playerPosX);

Assert.AreEqual(expected, actual, "The Box is being pushed into a another box");

}

// Check keystrokes are saved into an array

[TestMethod]

public void LevelKeystrokes\_CheckSaved()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " ";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

int actual = level.GetMoveHistory().Count;

Assert.AreEqual(expected, actual, "Moved history failed to save keystrokes");

}

// Ability to reverse last move(s)

[TestMethod]

public void LevelKeystrokes\_ReverseMove()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " ";

int expected = 0;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

level.UndoMove();

int actual = level.GetPlayerPosX();

Assert.AreEqual(expected, actual, "Failed to undo last move");

}

// Ability to reverse more then one move

[TestMethod]

public void LevelKeystrokes\_ReverseMoveMultiple()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 0;

int playerPosY = 0;

string levelDesign = " ";

int expected = 0;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

level.UndoMove();

level.UndoMove();

int actual = level.GetPlayerPosX();

Assert.AreEqual(expected, actual, "Failed to apply undo multiple times");

}

// Reverse more then the move history

[TestMethod]

public void LevelKeystrokes\_ReverseMoveMass()

{

int rowWidth = 3;

int rowCount = 1;

int playerPosX = 1;

int playerPosY = 0;

string levelDesign = " ";

int expected = 1;

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

level.MovePlayer(2); // 1 is up, 2 is right, 3 is down, 4 is left

level.UndoMove();

level.UndoMove();

int actual = level.GetPlayerPosX();

Assert.AreEqual(expected, actual, "Failed to stop undo when it should have");

}

// Ability to start game

[TestMethod]

public void GameControl\_StartGame()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "#### ####";

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

GameBoardView view = new GameBoardView();

GameController game = new GameController(view, level);

Assert.IsTrue(game.StartGame(), "Failed to start game");

}

// Ability to finish game

[TestMethod]

public void GameControl\_FinishGame()

{

int rowWidth = 3;

int rowCount = 3;

int playerPosX = 1;

int playerPosY = 1;

string levelDesign = "#### ####";

Level level = new Level(rowWidth, rowCount, playerPosX, playerPosY, levelDesign);

GameBoardView view = new GameBoardView();

GameController game = new GameController(view, level);

Assert.IsTrue(game.FinishGame(), "Failed to finish game");

}

// Check the file api handler loaded

[TestMethod]

public void FileAPI\_Exists()

{

GameFileAPI fileAPI = new GameFileAPI();

Assert.AreNotEqual(null, fileAPI, "Failed to load file handler api");

}

// Check api has all the required methods

[TestMethod]

public void FileAPI\_HasMethods()

{

GameFileAPI fileAPI = new GameFileAPI();

Assert.AreNotEqual(null, fileAPI.GetType().GetMethod("LoadLevels"), "Failed to load file handler api methods");

}

// Check the file api has found game files

[TestMethod]

public void FileAPI\_HasGameFiles()

{

GameFileAPI fileAPI = new GameFileAPI();

Assert.IsTrue(fileAPI.GetGameFiles(), "Failed to load file handler api game files");

}

}

}