Unit Testing

Function Name	`int getDateCode()`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	A profile was successfully created on April 1, 2024	profile name = “HELLO”	returns 4012024	returns 4012024	P
2	A profile was successfully created on March 30, 2024	profile name = “HEY”	returns 3302024	returns 3302024	P
3	A profile was successfully created on March 28, 2024	profile name = “SKRT”	returns 3282024	returns 3282024	P

Function Name	`void updateStatistics(struct Profile *CurrentProfile)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user won in classic game mode (easy)	profile: ANDRE The user won	profile: ANDRE EasyStats won increments and total seconds are updated from all games in the mode	The statistics in profile “ANDRE” are updated accordingly across all games won in easy	P

2	The user won in classic game mode (difficult)	profile: ANDRE	profile: ANDRE DifficultStats won increments and total seconds are updated from all games in the mode	The statistics in profile “ANDRE” are updated accordingly across all games won in difficult	P
3	The user won in custom game mode	profile: ANDRE	profile: ANDRE CustomStats won increments and total seconds are updated from all games in the mode	The statistics in profile “ANDRE” are updated accordingly across all games won in custom	P
4	The user lost in classic game mode (easy)	profile: ANDRE	profile: ANDRE EasyStats lost increments	The statistics in profile “ANDRE” are updated accordingly across all games won in easy	p
5	The user lost in classic game mode	profile: ANDRE	profile: ANDRE	The statistics	p

	(difficult)		DifficultStats lost increments	in profile “ANDRE” are updated accordingly across all games won in difficult	
6	The user lost in custom game mode	profile: ANDRE	profile: ANDRE CustomStats lost increments	The statistics in profile “ANDRE” are updated accordingly across all games won in custom	p

Function Name	`void updateRecentGames(struct Profile *CurrentProfile)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user has played 0 games	n/a	<profile name> has never played a game of minesweeper	<profile name> has never played a game of minesweeper	P
2	The user has played 1 game	The player either won, lost, or quit the	The board of the game will be updated as	The board of the game will be updated as	P

		game	Recent Game 1 containing its mode and outcome	Recent Game 1 together with its contents	
3	The user has played 2 games	The player either won, lost, or quit the game	The first game (Recent Game 1) will shift down into Recent Game 2 and the current game will become Recent Game 1	The first game of the player will shift down and the current game will be Recent Game 1	P
4	The user has played 3 games	The player either won, lost, or quit the game	The first game (Recent Game 2) will shift down to Recent Game 3. The second game (Recent Game 1) will shift down to Recent Game 2, and the current game will be updated on Recent Game 1	The first game of the player will shift down and the current game will be Recent Game 1	p

Function Name	`int getTileCode(int row, int column, struct Tile Board[][MAX_COLUMNS])`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	Getting the tile code of Board[1][1] which is a mine, is flagged, and is not revealed	row = 0 column = 0	returns 910	returns 910	P
2	Getting the tile code of Board[1][2] which is next to a mine, is not flagged, and is revealed	row = 0 column = 1	returns 101	returns 101	P
3	Getting the tile code of Board[4][4] which is surrounded by mines in all directions, is not flagged, and is not revealed	row = 3 column = 3	returns 800	returns 800	P

Function Name	`void updateProfile(struct Profile *CurrentProfile)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The profile's text file does not exist.	The profile exists in the list of profiles but the text file does not.	The profile's test file will be created and updated accordingly.	The profile's test file was created and updated accordingly.	P
2	The profile's text file exists and contains	The profile and	The profile's	The profile's	P

	outdated information.	its text files exists and the profile had just concluded a game.	test file will be updated accordingly.	test file was updated accordingly.	
3	The profile only has 1 recent game after a game concluded.	The profile's text file will be updated but will only contain information pertaining to recent game 1, besides their basic information.	The profile's text file only contains recent game 1 information after update.	The profile's text file only contained recent game 1 information after update.	P

Function Name	`void initializeProfile(struct Profile *CurrentProfile, char name[])`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The profile being initialized is a new one being created.	The profile instance and the name of the new profile.	The profile information and statistics have been reset and the creation date maintained	The profile information and statistics have been reset and the creation date maintained	P

			(today).	(today).	
2	The profile being initialized is the default guest profile.	The profile instance and “GUEST”.	The profile information and statistics have been reset and the creation date maintained (today).	The profile information and statistics have been reset and the creation date maintained (today).	P
3	The profile being created is an old profile being reset.	The profile instance and the name of the old profile.	The profile information and statistics have been reset and the creation date maintained (old date).	The profile information and statistics have been reset and the creation date maintained (old date).	P

Function Name	`void loadLeaderboard(struct Leaderboard *CurrentLeaderboard)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The leaderboard has easy records.	The leaderboard struct instance	The leaderboard’s easy records in the text file are loaded into the	The leaderboard’s easy records in the text file are loaded	P

			leaderboard struct.	into the leaderboard struct.	
2	The leaderboard has difficult records.	The leaderboard struct instance	The leaderboard's difficult records in the text file are loaded into the leaderboard struct.	The leaderboard's difficult records in the text file are loaded into the leaderboard struct.	P
3	The leaderboard has custom records.	The leaderboard struct instance	The leaderboard's custom records in the text file are loaded into the leaderboard struct.	The leaderboard's custom records in the text file are loaded into the leaderboard struct.	P

Function Name	`void sortProfileNames(string20 ProfileNames[], int n)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	There are zero elements in the array.	[ ]	The function terminates and will not mutate	The function terminated and did not	P

			the array.	mutate the array.	
2	There are two elements in the array, in descending alphabetical order.	[“RAIDEN”, “ACHERON”]	The profile names will be swapped and reflected in the array.	The profile names were swapped and reflected in the array.	P
3	There are MAX_PROFILES (10) profiles in the array, in randomized order.	[“AAA”, “GGG”, “BBB”, “III”, “EEE”, “DDD”, “FFF”, “JJJ”, “CCC”, “HHH”]	The profile names will be arranged in alphabetical order and reflected in the array.	The profile names were arranged in alphabetical order and reflected in the array.	P

Function Name	`int getProfileNames(string20 ProfileNames[])`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	Profile directory contains: 0 profiles	No profiles currently exist	0 (and profiles are reflected in the array ProfileNames)	0 (and profiles are reflected in the array ProfileNames)	P
2	Profile directory contains: 1 profile	1 profile currently exists • HELLO	1 (and profiles are reflected in the array ProfileNames)	1 (and profiles are reflected in the array ProfileNames)	P

3	Profile directory contains: 3 profiles	3 profiles currently exists • HELLO • HEY • HALO	3 (and profiles are reflected in the array ProfileNames)	3 (and profiles are reflected in the array ProfileNames)	P

Function Name	`void toUpperCaseString(string20 string)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	String contains all lowercase letters	“string”	“STRING”	“STRING”	P
2	String contains all uppercase letters	“STRING”	“STRING”	“STRING”	P
3	String contains a mixture of lowercase and uppercase letters	“stRiNG”	“STRING”	“STRING”	P

Function Name	`int profileExists(string20 name)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The profile name exists.	“RAIDEN”	1	1	P

2	The profile name does not exist.	“EI”	0	0	P
3	The profile name exists but is not capitalized throughout.	“RAidEn”	1	1	P

Function Name	`void loadProfile(struct Profile *CurrentProfile, string20 name)`

#	Test Description	Test Input	Expected Result	Actual Result	Pass / Fail (P/F)
1	The profile exists and they have their corresponding text file.	The current profile and their name.	The current profile struct will be updated with respect to the profile’s text file.	The current profile struct was updated with respect to the profile’s text file.	P
3	The profile exists but the text file does not.	The current profile and their name.	A new text file will be created for the current profile and initialized.	A new text file was created for the current profile and initialized.	P

Function Name	`void selectProfile(struct Profile *CurrentProfile)`

#	Test Description	Test Input	Expected Result	Actual Result	Pass / Fail (P/F)
1	The user entered 0	profile = “0”	Returns back to menu with the current profile as is.	Returns back to menu with the current profile as is.	P
2	The user entered a non-existent profile	Assume the ff exists: • HELLO profile = “HEY”	The profile '<profile name>' does not exist.	The profile '<profile name>' does not exist.	P
3	The user entered an existing profile	Assume the ff exists: • HELLO profile = “HELLO”	Successful. The current profile has been changed from <previous name> to <chosen name>	Successful. The current profile has been changed from <previous name> to <chosen name>	P

Function Name	`int isValidProfileName(string20 name)`				
#	Test Description	Test Input	Expected Result	Actual Result	Pass / Fail (P/F)
1	User entered a profile name less than 3 characters	name = “hi”	The name must contain	The name must contain	P

			between 3 to 20 characters!	between 3 to 20 characters!	
2	User entered a profile name containing characters not from the English Alphabet	name = “@hi”	The name must only contain letters from the English alphabet	The name must only contain letters from the English alphabet	P
3	User entered a valid profile name, however it already exists Assume the ff profile exists: ● HELLO	name = “HELLO”	The name '%s' has already been taken. Please provide another name!	The name '%s' has already been taken. Please provide another name!	P
4	User entered “GUEST” as a profile name	name = “GUEST”	The name '%s' has already been reserved for the program. Please provide another name!	The name '%s' has already been reserved for the program. Please provide another name!	P

Function Name	`int createProfile(struct Profile *CurrentProfile, int theme)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	name contains characters greater than	name =	The profile	The profile	P

	maximum amount (20)	“BaalRaidenShogunAndYaeMiko”	name must contain between 3 to 20 characters! Returns 1 if re-entered name is valid	name must contain between 3 to 20 characters! Returns 1 if re-entered name is valid	
2	name is within the 3-20 character range, but contains characters not from the alphabet	name = “R@iden”	The name must only contain letters from the English alphabet Returns 1 if re-entered name is valid	The name must only contain letters from the English alphabet Returns 1 if re-entered name is valid	P
3	name already exists/taken	Assume that the following profiles already exist: ● RAIDEN ● YAE ● NAHI name = “NAHI”	The name must not have already been taken by an existing profile	The name must not have already been taken by an existing profile	P
4	User made a valid profile name	Assume that the following profiles already exist: ● RAIDEN ● YAE	Successful. The profile ‘BAAL’ has been created	Successful. The profile ‘BAAL’ has been created	P

		• NAHI name = “BAAL”	returns 1	returns 1	

Function Name	`int confirmAction()`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user wants to confirm (Yes)	c = ‘Y’	returns 1	returns 1	P
2	The user wants to decline (No)	c = ‘N’	returns 0	returns 0	P
3	The user did not enter a valid input	c = ‘K’	Prompts the user to make a valid input	Yes [Y] or No [N]:	P

Function Name	`void deleteProfile(string20 name, int theme)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user wants to delete an existing profile Assume the ff profiles exists: • HELLO • HEY	profile = “HELLO” c = ‘Y’	Are you sure you want to delete the profile 'HELLO'?	Successful. The profile “HELLO” has been deleted.	P

			The said profile will be deleted		
2	The user wants to delete a non-existent profile Assume the ff profiles exists: ● HELLO ● HEY	profile = “HALO”	The profile “HALO” does not exist	The profile “HALO” does not exist	P
3	The user initially wants to delete a profile, but changed their mind	profile = “HELLO” c = ‘N’	Are you sure you want to delete the profile 'HELLO'? Returns to the menu	Returns to main menu	P

Function Name	`void profileHandler()`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user wants to select an existing profile Assume the ff profiles exists: ● HELLO ● HEY	userResponse = ‘a’ name = HELLO	Successful. The current profile has been changed from <previous	Successful. The current profile has been changed from <previous	P

			name> to <chosen name>	name> to <chosen name>	
2	The user wants to create a new profile Assume the ff profiles exists: • HELLO • HEY	userResponse = 'b' name = HALO	Successful. The profile 'HALO' has been created	Successful. The profile 'HALO' has been created	P
3	The user wants to delete and existing profile Assume the ff profiles exists: • HELLO • HEY	userResponse = 'c' profile = HELLO	Successful. The profile 'HELLO' has been deleted	Successful. The profile 'HELLO' has been deleted	P

Function Name	`int getRandInt(int min, int max)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user wants to generate an integer between 1 to 1	min = 1 max = 1	1	1	P
2	The user wants to generate an integer between 1 to 101	min = 1 max = 101	[1, 101]	14	P
3	The user wants to generate an integer between 1 to 10101	min = 1 max = 10101	[1, 10101]	2273	P

Function Name	`void generateClassicGame(struct Tile Board[][15], int mineLocations[], int rows, int columns, int numMines)`

#	Test Description	Test Input	Expected Result	Actual Result	Pass / Fail (P/F)
1	An easy game is being generated.	rows = 8 columns = 8 numMines = 10	The board will be filled with 10 randomized mines and the tile members are updated accordingly.	The board is filled with 10 randomized mines and the tile members are updated accordingly.	P
2	A difficult game is being generated.	rows = 10 columns = 15 numMines = 35	The board will be filled with 35 randomized mines and the tile members are updated accordingly.	The board is filled with 35 randomized mines and the tile members are updated accordingly.	P

Function Name	`void generateCustomGame(struct Tile Board[][15], int *rows, int *columns, int mineLocations[], int *numMines, int theme)`

#	Test Description	Test Input	Expected Result	Actual Result	Pass / Fail (P/F)

1	User entered an existing level Assume the ff levels exist: • LVL	name = “LVL”	return 1	return 1	P
2	The user entered a non-existing level Assume the ff levels exist: • LVL	name = “SUPER”	The level 'SUPER' does not exist!	The level 'SUPER' does not exist!	P
3	The user wants to go back to menu	name = “0”	return 0	return 0	P

Function Name	`void incrementTileState(struct Tile Board[][15], int row, int column, int numRows, int numColumns)`

#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The row/column is out of bounds.	row = -1 column = 1	Nothing happens.	Nothing happens.	P
2	The tile is valid but is a mine.	row = 1 column = 1 Board[1][1].state = 9	Nothing happens.	Nothing happens.	P
3	The tile is valid and is not a mine.	row = 1 column = 2 Board[1][2].state =	Board[1][2].state = 1	Board[1][2].state = 1	P

		0			

Function Name	`void initializeTileStates(struct Tile Board[10][15], int mineLocations[], int rows, int columns, int numMines)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	Mines have been randomized around the board.	Mines have been randomized around the board.	Non-mine tiles' states reflect the number of mines adjacent to them.	Non-mine tiles' states reflect the number of mines adjacent to them.	P

Function Name	`Int detectKeyPress(int *currRow, int *currColumn, int rows, int columns)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user wants to hover to the left and is not on the border	currColumn = 1	return 75 currColumn = 0	return 75 currColumn = 0	P

2	The user wants to hover to the right but is on border	currColumn = columns - 1	return 77 currColumn = columns - 1	return 77 currColumn = columns - 1	P
3	The user wants to hover up but is on border	currRow = 0	return 72 currRow = 0	return 72 currRow = 0	P
4	The user wants to hover down and is not on the border	currRow = 2	return 80 currRow = 3	return 80 currRow = 3	P
5	The user either wants to inspect, flag, remove flag, unselect, or quit, so they have to enter the enter key	currRow = 1 currColumn = 1	return 13 currRow = 1 currColumn = 1	return 13 currRow = 1 currColumn = 1	P

Function Name	`void revealTiles(struct Tile Board[][15], int rows, int columns, int row, int column)`

#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The tile being revealed is out of bounds, i.e., invalid row/column.	row = -1 column = 1	return; nothing happens	return; nothing happens	P
2	The tile being revealed has already been	row = 1	return;	return;	P

	revealed.	column = 1 Board[1][1].isRevealed = 1	nothing happens	nothing happens	
3	The tile is within bounds, unrevealed and is a numbered tile, i.e., has a state between 1 to 8	row = 1 column = 1 Board[1][1].isRevealed = 0 Board[1][1].state = 4	Board[1][1].isRevealed = 1	Board[1][1].isRevealed = 1	P
4	The tile is within bounds, unrevealed and is a blank tile, i.e., has a state of 0	row = 1 column = 1 Board[1][1].isRevealed = 0 Board[1][1].state = 4	Board[1][1].isRevealed = 1 the recursive code gets triggered; revealTiles get evoked on all 8 adjacent tiles	Board[1][1].isRevealed = 1 the recursive code gets triggered; revealTiles get evoked on all 8 adjacent tiles	p

Function Name	`Int getGameState(struct Tile Board[][15], int rows, int columns, int mineLocations[], int numMines)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The player inspected a mine	Assume a mine is revealed	2	2	P
2	The player won	All non-mine tiles have been revealed	1	1	P
3	The game is ongoing	A mine has not yet been revealed and not all non-mine tiles have been revealed	0	0	P

Function Name	`void setMineVisibility(int visibility, struct Tile Board[][15], int mineLocations[], int numMines)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**

1	The user wants to reveal all mines.	visibility = 1	All mines will be revealed in next printing.	All mines were revealed in next printing.	P
2	The user wants to hide all mines.	visibility = 0	All mines will be hidden in next printing.	All mines were hidden in next printing.	P

Function Name	`Int updateRecord(string20 name, int time, string20 names[], int times[], int *numRecords)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The player's game set a new record.	name = "EI" time = 2 Current leaderboard arrays: 2 YAE 4 MIKO 6	Leaderboard arrays after update: 3 EI 2 YAE 4 MIKO 6 return 1	Leaderboard arrays after update: 3 EI 2 YAE 4 MIKO 6 return 1	P
2	The player's game did not set a new record / it went beyond the maximum number of records.	name = "EI" time = 14 Current leaderboard arrays: 10	Leaderboard arrays after update: 10 YAE 4 MIKO 5 YAE 6	Leaderboard arrays after update: 10 YAE 4 MIKO 5 YAE 6	P

		YAE 4 MIKO 5 YAE 6 MIKO 7 YAE 8 MIKO 9 YAE 10 MIKO 11 YAE 12 MIKO 13	MIKO 7 YAE 8 MIKO 9 YAE 10 MIKO 11 YAE 12 MIKO 13	MIKO 7 YAE 8 MIKO 9 YAE 10 MIKO 11 YAE 12 MIKO 13	

Function Name	`Int updateLeaderboard(string20 mode, string20 outcome, string20 name, int seconds, struct Leaderboard *CurrentLeaderboard)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	Assume the ff: -— CLassic -> Easy — Name | Record 1.) HELLO | 432 seconds . . . . The user won a game in classic (easy) with a time less than the first place	name = HEY (The user won the game) seconds = 123	Prints the ff. In the leaderboard text file: 2 HEY 123 HELLO 432 0 0 returns rank = 1	Prints the ff. In the leaderboard text file: 2 HEY 123 HELLO 432 0 0 returns rank = 1	P

2	Assume the ff: -— CLassic -> Difficult — Name | Record 2.) HELLO | 700 seconds . . . . The user won a game with the same profile, however its time is greater than the existing record in the leaderboard.	profile = HELLO (The user won the game) seconds = 888	Prints the ff. In the leaderboard text file: 0 2 HELLO 700 HELLO 888 0 returns rank = 2	Prints the ff. In the leaderboard text file: 0 2 HELLO 700 HELLO 888 0 returns rank = 2	P
3	Assume the ff: -— CLassic -> Custom — Name | Record 3.) HELLO | 8 seconds . . . . The user won a game in custom mode with a different profile but with the same time with the current record in the leaderboard	profile = BETTER (The user won the custom game) seconds = 8	Prints the ff. In the leaderboard text file: 0 0 2 HELLO 8 HELLO 8 returns rank = 2	Prints the ff. In the leaderboard text file: 0 0 2 HELLO 8 HELLO 8 returns rank = 2	P

Function Name	`void gameHandler(struct Profile *CurrentProfile, struct Leaderboard *CurrentLeaderboard)`				
#	**Test Description**	**Test**	**Expected**	**Actual**	**Pass / Fail**

	(PRE-GAME)	**Input**	**Result**	**Result**	**(P/F)**
1	The user want to play a classic - easy game	userResponse = ‘a’ *(classic)* userResponse = ‘a’ *(easy)*	Output the board using correct function calls with a classic-easy game format	Successfully output the board format	P
2	The user want to play a classic - difficult game	userResponse = ‘a’ *(classic)* userResponse = ‘b’ *(difficult)*	Output the board using correct function calls with a classic-difficult game format	Successfully output the board format	P
3	The user want to play a custom game	userResponse = ‘b’	Outputs the board using the correct function calls assuming that the file is valid	Successfully output the board format	P
#	**Test Description** ***(GAME ONGOING)***	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user want to play a classic - easy game and decided to quit midway	Assume user used arrow keys until enter key userResponse = ‘Q’	gameState = 3 (user quit)	The user successfully quitted the game	P

		userResponse = ‘Y’			
2	The user want to play a classic - difficult game	Assume user used arrow keys until enter key userResponse = ‘F’ (enter key) userResponse = ‘R’	The state of the tiles of the board remain the same gameState = 0 (game ongoing)	The game is still ongoing	P
3	The user want to play a custom game	Assume user used arrow keys until enter key userResponse = ‘F’ userResponse = ‘U’	The state of the tiles of the board remain the same gameState = 0 (game ongoing)	The game is still ongoing	P
#	**Test Description *(POST-GAME)***	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user want to play a classic - easy game	N/A (users has quit)	Game quitted successfully	Game quitted successfully	P
2	The user want to play a classic - difficult	Assume the user	Winning ASCIIi	YOU WON	P

	game	revealed all non-mine tiles	displays		
3	The user want to play a custom game	Assume that at some point, the user inspected a mine	Losing ASCII displays	YOU LOST You inspected a mine!	P

Function Name	`int isValidLevel(struct Tile Board[][15], int rows, int columns)`			
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The level has no mines	rows = 5 columns = 7 The user finished editing the level without placing mines	0	0	P
2	The level only has no blank tiles	rows = 7 columns = 7 The user places mines on all tiles	0	0	P

3	The level is valid (not all tiles are mines)	rows = 9 columns = 15 Assume that the user places one mine at row and column (1, 1)	1	1	P

Function Name	`void createLevel(int theme)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	(Assume there are already 10 existing levels) The user wants to create a new level	n/a (user won't be able to input)	Sorry, you cannot create a new level as there can only be a maximum of %d levels.	Sorry, you cannot create a new level as there can only be a maximum of %d levels.	P
2	Assume the ff. Levels exists: • SUPER The user wants to create a new level	name = @ numRows = 11 numColumns = 11 userRespone = ‘P’	The user will be able to successfully create a level The level will be a 11 by 11 grid with only 1 mine at the	Successful. The custom level '@' has been saved.	P

		row = 1 column = 1 userResponse = 'F' c = 'Y'	upper left corner (1, 1)		
3	Assume the ff. Levels exists: ● SUPER The user wants to create a new level	name = SUPER	The name 'SUPER' has already been taken!	The name 'SUPER' has already been taken!	P

Function Name	`Int deleteLevel(int theme)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	Assume the ff. Levels exists: ● SUPER The user wants to delete a level, but it does not exist	name = "SUPPERR"	The level 'SUPERR' does not exist!	The level 'SUPERR' does not exist!	P
2	Assume the ff. Levels exists: ● SUPER The user wants to delete a level	name = "SUPER" c = 'Y'	Successful. The custom level 'SUPER' has been deleted from the levels folder.	Successful. The custom level 'SUPER' has been deleted from the levels folder.	P

3	The user wishes to go back to menu	name = “0”	The user returns to the main menu	The user returns to the main menu	P

Function Name	`float getWinRate(int wins, int losses)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	wins is greater than losses and are non-negative integers	wins = 5 losses = 4	55.56	55.56	P
2	wins is less than losses and are non-negative integers	wins = 3 losses = 5	37.50	37.50	P
3	wins equal to losses and are positive integers	wins = 3 losses = 3	50.00	50.00	P
4	wins is zero and losses is a non-negative integer	wins = 0 losses = 5	0.00	0.00	P
5	wins and losses are zero	wins = 0 losses = 0	0.00	0.00	

Function Name	`float getAverageSeconds(int totalSeconds, int gamesWon)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**

1	There are no games won, hence no total seconds	gamesWon = 0 totalSeconds = 0	0	0	P
2	gamesWon and totalSeconds are non-negative integers	gamesWon = 4 totalSeconds = 360	90.00	90.00	P
3	gamesWon and totalSeconds are non-negative integers and they are equal	gamesWon = 1 totalSeconds = 1	1.00	1.00	P

Function Name	`void statisticsScreen(struct Profile *CurrentProfile, int theme)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	Assume the ff. Information is for the profile: HEY and the profile exists	Assume that profile "HEY": • has played a lifetime games of: 1 • Won in custom with 8 seconds • The profile was created on March 30, 2024	The ff information is expected to be reflected: ---- Basic Profile Information ----- Name: HEY Date of Profile Creation: 03/30/2024 Lifetime Games Played: 1	The information from profile "HEY" was properly reflected in the statistics screen with the board reflected	P

			. . . . ----- Custom Game Statistics ----- Custom Win Rate: 100.00 % Custom Games Won Average Time: 8 seconds Custom Games Won: 1 Custom Games Lost: 0 (Game 1 Board Reflects)		
2	Assume the ff. Information is for the profile: HELLO and the profile exists	Assume that profile “HELLO”: ● has played a lifetime games of: 5 ● Won in classic (all 5 games are easy) with a total of 750 seconds ● The profile was created on	----- Basic Profile Information ----- Name: HEY Date of Profile Creation: 03/30/2024 Lifetime Games Played: 1 ----- Classic	The information from profile “HELLO” was properly reflected in the statistics screen while also reflecting the three recent games	P

		March 30, 2024	Game Statistics --- Easy Win Rate: 0.00 % Easy Games Won Average Time: 0 seconds Easy Games Won: 0 Easy Games Lost: 0 Difficult Win Rate: 0.00 % Difficult Games Won Average Time: 0 seconds Difficult Games Won: 0 Difficult Games Lost: 0 . . . . (Game 5 Board Reflects) (Game 4 Board Reflects) (Game 3 Board Reflects)		

3	Assume the ff. Information is for the profile: SKRRT and the profile exists	Assume that profile “HELLO”: • The user just created the profile and has not played any games • The profile was created on March 30, 2024	The statistics of the profile will all contain “0” except for Date of Profile Creation	The statistics of the profile contains “0” except for Date of Profile Creation	P

Function Name	`void leaderboardScreen(struct Leaderboard *CurrentLeaderboard, int theme)`				
#	**Test Description**	**Test Input**	**Expected Result**	**Actual Result**	**Pass / Fail (P/F)**
1	The user wants to delete the all-time-leaderboard	getch() == 224 and getch() == 83 c = ‘Y’	Successful. The all-time leaderboard has been reset.	The current information in the all-time-leader board is resetted.	P
2	The user wants to delete the all-time-leaderboard, but decided not to execute it	etch() == 224 and getch() == 83 c = ‘N’	The leaderboard is not affected	The leaderboard is not affected	P
3	The user wishes to back to the main	getch() != 224	The user went	The user went	P

	menu	and getch() != 83	back to the menu screen	back to the menu screen	