PROGRAM TEST SCRIPT, ROMAN - S19B

1.) initializeVariables();

Function Description:

- The 'intializeVariables' function initializes the necessary pre-game variables.

Prototype:

```
void initializeVariables(int *nPoints, int *nStatus, int *nPowerPosition,
int *nRounds, int *nPowerClaimed, int *nMultiplier, int *nPlayerPosition,
int *nEnemyPosition_1, int *nEnemyPosition_2,int *nEnemyPosition_3, int
*nEnemyCycle_1, int *nEnemyCycle_2, int *nEnemyCycle_3);
```

Pre-Condition:

- All variables are integer-type.

Return:

- This function does not return anything; void.

#	Test Case Description	Input Data	Expected Output	Actual Output	Pass / Fail

1	Uninitialized variables.	Garbage values (since the variables passed into the function are uninitialized)	*nPoints = 0;
			*nStatus = 0;
			*nPowerPosition = 0;
			*nRounds = 1;
			*nPowerClaimed = 0;
			*nMultiplier = 1;
			*nPlayerPosition = 1;
			*nEnemyPosition_1 = rand() % 4 + 1;
			*nEnemyPosition_2 = rand() % 4 + 1;
			*nEnemyPosition_3 = rand() % 4 + 1;
			*nEnemyCycle_1 = 1;
			*nEnemyCycle_2 = 1;
			*nEnemyCycle_3 = 1;
				*nPoints = 0;
				*nStatus = 0;
				*nPowerPosition = 0;
				*nRounds = 1;
				*nPowerClaimed = 0;
				*nMultiplier = 1;
				*nPlayerPosition = 1;
				*nEnemyPosition_1 = rand() % 4 + 1;
				*nEnemyPosition_2 = rand() % 4 + 1;
				*nEnemyPosition_3 = rand() % 4 + 1;
				*nEnemyCycle_1 = 1;
				*nEnemyCycle_2 = 1;
				*nEnemyCycle_3 = 1;
					Pass

2	Initialized variables.	Initialized game variables. (i.e nPoints = 5, nStatus = 5, etc.)	*nPoints = 0;
			*nStatus = 0;
			*nPowerPosition = 0;
			*nRounds = 1;
			*nPowerClaimed = 0;
			*nMultiplier = 1;
				*nPoints = 0;
				*nStatus = 0;
				*nPowerPosition = 0;
				*nRounds = 1;
				*nPowerClaimed = 0;
				*nMultiplier = 1;
					Pass

			*nPlayerPosition = 1; *nEnemyPosition_1 = rand() % 4 + 1; *nEnemyPosition_2 = rand() % 4 + 1; *nEnemyPosition_3 = rand() % 4 + 1; *nEnemyCycle_1 = 1; *nEnemyCycle_2 = 1; *nEnemyCycle_3 = 1;	*nPlayerPosition = 1; *nEnemyPosition_1 = rand() % 4 + 1; *nEnemyPosition_2 = rand() % 4 + 1; *nEnemyPosition_3 = rand() % 4 + 1; *nEnemyCycle_1 = 1; *nEnemyCycle_2 = 1; *nEnemyCycle_3 = 1;	
3	Missing variable declaration.	All variables except for nPlayerPosition ..	Compiler Error	Compiler Error	Fail

- Note that the expected output / actual output in this function refers to the initialized outcome of the variables, NOT the function's return output.

2.) printField();

Function Description:

- The 'printField' function produces a grid divided into two portions; the enemy field and the player field. The enemy field is 5x4 (5 rows, 4 columns) and the player field is 1x4 (1 row, 4 columns). Note that the power-up spawns in the enemy field.

Prototype:

```
void printField(int nPlayerPosition, int nEnemyPosition_1, int
nEnemyPosition_2, int nEnemyPosition_3, int nPowerPosition);
```

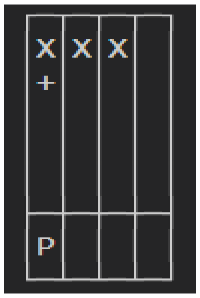
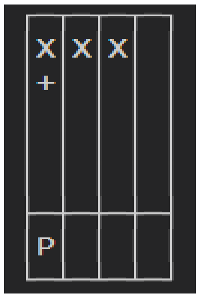
Pre-Condition:

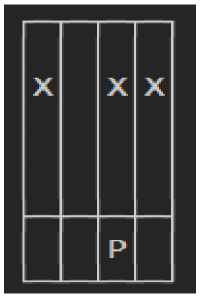
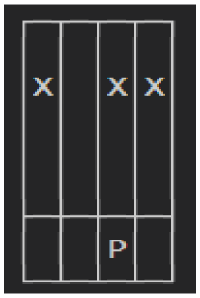
- Enemy and power-up variables are non-negative values. The player's position value must be between 1 or 4 inclusive.

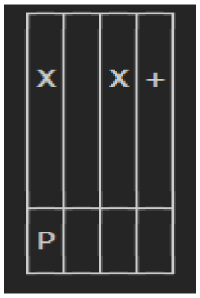
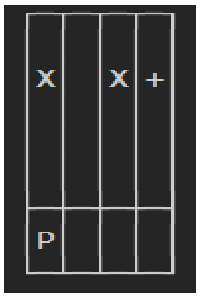
Return:

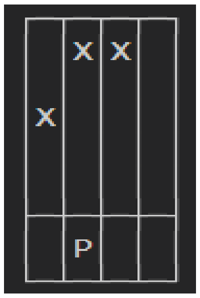
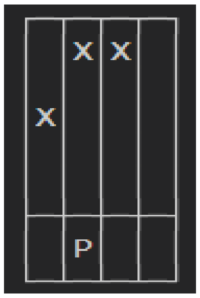
- This function does not return anything; void.

#	Test Case Description	Input Data	Expected Output	Actual Output	Pass / Fail

1	All variable values are positive with differing enemy & power-up variable values.	nPlayerPosition = 1;
		nEnemyPosition_1 = 1;
		nEnemyPosition_2 = 2;
		nEnemyPosition_3 = 3;
		nPowerPosition = 5;
					Pass

2	Power-Up variable value is not between 1 to 20.	nPlayerPosition = 3;
		nEnemyPosition_1 = 5;
		nEnemyPosition_2 = 7;
		nEnemyPosition_3 = 8;
		nPowerPosition = 0;
					Pass
					(Note that this scenario happens when power-up is hit or because it hasn't spawned in yet)

3	Enemy position values are similar.
	(Meaning they are in the same grid position)
		nPlayerPosition = 1;
		nEnemyPosition_1 = 5;
		nEnemyPosition_2 = 5;
		nEnemyPosition_3 = 7;
		nPowerPosition = 8;
					Pass

4	Power-Up variable value is the same as the enemy's position variable value.	nPlayerPosition = 2;
		nEnemyPosition_1 = 3;
		nEnemyPosition_2 = 2;
		nEnemyPosition_3 = 9;
		nPowerPosition = 9;
					Pass
					(The game instructions recognize this as a valid outcome; enemy destroys the power-up)

- Note that the expected output / actual output in this function refers to the outcome (whether through printing or changing variable values) produced by the function, NOT the function's return output.

3.) inputAction();

Function Description:

- The ‘inputAction’ function prompts the user for input. If the input is movement based, the player adjusts accordingly.
-

Prototype:

```
void inputAction(int *nPlayerAction, int *nPlayerPosition);
```

Pre-Condition:

- *nPlayerPosition’s value must be in between 1 and 4 inclusive. *nPlayerAction is unassigned pre-function or contains the previous nPlayerAction value (after the first round).

Return:

- This function does not return anything; void.

#	Test Case Description	Input Data	Expected Output	Actual Output	Pass / Fail
1	Valid nPlayerPosition value & valid nPlayerAction input.	Pre-Function Value: nPlayerPosition = 1; Player inputs 2 as the player action in the function: *nPlayerAction = 2;	PlayerPosition updates to 2; *(player moves right)*	PlayerPosition updates to 2; *(player moves right)*	Pass
2	Valid nPlayerPosition value & invalid nPlayerAction input.	Pre-Function Value: nPlayerPosition = 1; Player inputs 10 as the player action in the function: *nPlayerAction = 10;	Error statement appears in the terminal: “Invalid Input”	Error statement appears in the terminal: “Invalid Input”	Pass
3	nPlayerPosition is either in the rightmost/leftmost (1 or 4) column & player inputs invalid move.	Pre-Function Value: nPlayerPosition = 4; Player inputs 10 as the player action in the function: *nPlayerAction = 2;	Error statement appears in the terminal: “Out of Bounds”	Error statement appears in the terminal: “Out of Bounds”	Pass

- Note that the expected output / actual output in this function refers to the outcome (whether through printing or changing variable values) produced by the function, NOT the function's return output. Moreover, the nPlayerAction input data is scanned IN the function while the nPlayerPosition is obtained OUTSIDE the function *(which is then updated in the function.)*

4.) playerShoot();

Function Description:

- The 'playerShoot' function is activated if nAction is 3. Furthermore, this function updates the necessary variables and calculates the proper response if the player shoots an enemy or power-up.

Prototype:

```
void playerShoot(int *nPlayerShoot, int *nPlayerPosition, int
*nEnemyPosition_1, int *nEnemyPosition_2, int *nEnemyPosition_3, int
*nEnemyCycle_1, int *nEnemyCycle_2, int *nEnemyCycle_3, int
*nEnemyDeath, int *nPowerClaimed, int *nPowerPosition, int
*nRoundsLeft, int *nPoints, int *nMultiplier);
```

Pre-Condition:

- All parameters are non-negative and nAction is 3.

Return:

- This function does not return anything; void.

#	Test Case Description	Input Data	Expected Output	Actual Output	Pass / Fail
1	Player shoots an enemy with another enemy behind it.	nPlayerPosition = 2; nEnemyPosition_1 = 2; nEnemyPosition_2 = 6; nEnemyPosition_3 = 8;	Only enemy 3 dies.	Same as expected output.	Pass

2	Player shoots a power-up with an enemy behind it.	nPlayerPosition = 2; nEnemyPosition_1 = 2; nPowerPosition = 6;	The player should obtain the power-up (which is to be applied in the following round) and kill the enemy.	The player should obtain the power-up (which is to be applied in the following round) and kill the enemy.	Pass *(The game instructions recognize this as a valid outcome)*
3	Player shoots an enemy with the power-up applied.	nPlayerPosition = 2; nEnemyPosition_1 = 2; nPowerClaimed = 1;	A 2x multiplier is applied, the player should get 20 points.	A 2x multiplier is applied, the player should get 20 points.	Pass

- Note that the expected output / actual output in this function refers to the outcome (whether through printing or changing variable values) produced by the function, NOT the function's return output.

5.) updateEnemyPosition();

Function Description:

- The 'updateEnemyPosition' function updates the enemy position depending on their cycle.
- An enemy's cycle refers to their level in the provided algorithm. For example, cycle 1 means the enemy moves right, 2 or 4 means enemy moves down, and 3 means the enemy moves left.

Prototype:

```
void updateEnemyPosition(int *nEnemy, int *nEnemyCycle);
```

Pre-Condition:

- nEnemy and nEnemyCycle variables are non-negative values.

Return:

- This function does not return anything; void.

#	Test Case Description	Input Data	Expected Output	Actual Output	Pass / Fail

1	Regular enemy movement.	*nEnemy = 2;
		*nEnemyCycle = 2;
			Enemy moves to the expected grid.	Enemy moves to the expected grid.	Pass

2	nEnemy's cycle requires the enemy to move past its borders.	*nEnemy = 4;
		*nEnemyCycle = 1;
			Enemy moves down instead.	Enemy moves down instead.	Pass

3	nEnemy moves down while they are in the bottom enemy row. (row 5)	*nEnemy = 20;
		*nEnemyCycle = 2;
			Enemy moves down and disappears.
			
			(Player is defeated in this scenario)
				Enemy moves down and disappears.
				
				(Player is defeated in this scenario)
					Pass

- Note that the expected output / actual output in this function refers to the outcome (whether through printing or changing variable values) produced by the function, NOT the function's return output.

6.) updatePowerUp();

Function Description:

- The 'updatePowerUp' function updates the position of the power-up (represented as the character '+' / the plus sign). Moreover, the function covers cases when the player gains the power up.

Prototype:

```
void updatePowerUp(int *nRounds, int *nPowerPosition, int
*nEnemyPosition_1, int *nEnemyPosition_2, int *nEnemyPosition_3, int
*nPowerClaimed, int *nMultiplier, int *nRoundsLeft);
```

Pre-Condition:

- All parameters are non-negative.

Return:

- This function does not return anything; void.

#	Test Case Description	Input Data	Expected Output	Actual Output	Pass / Fail
1	Power-up spawns.	*nRounds = 5;	Power-up spawns randomly in the enemy field grid.	Power-up spawns randomly in the enemy field grid.	Pass
2	Power-up is claimed.	nPowerClaimed >= 1;	A 2x multiplier is applied and the rounds left *(of claiming the power-up)* decrements each round.	A 2x multiplier is applied and the rounds left *(of claiming the power-up)* decrements each round.	Pass

3	nEnemy moves to where a power-up is located.	*nEnemy = 1;
		*nEnemyCycle = 1;
		
		nPowerPosition = 2;
			Enemy moves and destroys the power-up.	Enemy moves and destroys the power-up.	Pass

4	Rounds left of having the power-up is decremented to zero.	*nRoundsLeft is decremented to zero. *nRoundsLeft = 0;	Multiplier is back to 1 and the power-up is removed from the player.	Multiplier is back to 1 and the power-up is removed from the player.	Pass

- Note that the expected output / actual output in this function refers to the outcome (whether through printing or changing variable values) produced by the function, NOT the function's return output.