CS 202 - S22 - Assignment 5

Mr. Piotrowski

March 2022

1 Introduction

Welcome to assignment 5, the assignment designed to help you practice **virtuals**. This is another 2-dimensional game (*similar to battle chess, but not as complicated*). In this game the player mines through a 2D grid full of rocks, treasure, armor, and carnivorous silverfish!

2 Program Flow

The program begins by asking for a seed, grid height, grid width, and difficulty level.

```
[piotrj1@bobby test]$ ./a.out
Enter Seed: 3
Enter Height of Game Grid (must be 10+):
10
Enter Width of Game Grid (must be 10+):
10
Enter the Difficulty Level (must be 1+): 1
```

Next, the player is presented with the grid. The grid represents a mine full of rocks (*the '0' chars*). The player is the 'M' character. The player is allowed to move up, left, right, and down.

```
------------------
| Current Score: |  0
------------------
 * Armor Bonus: *  0

|#|#|#|#|#|#|#|#|#|#|
|#|M|0|0|$|^|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|^|0|0|0|0|#|
|#|0|0|0|0|$|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|0|0|^|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|^|$|0|0|0|0|$|E|
|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
```

When the player moves, the miner will attempt to "destroy" the object in the direction chosen. The miner could run into one of the 7 objects:

- rock (*'0' characters*)
- hardened_wall (*'#' characters*)
- spike_armor (*'^' characters*)
- hidden_treasure (*'$' characters*)
- empty_space (*' ' characters*)

- mine_exit (*'E' characters*)
- carnivorous_silverfish (*'0' characters*)

As the miner interacts with the objects, different messages will be printed. If the miner is able to destroy the object without any problems, the miner will move into the next spot.

```
r
The miner digs through some ordinary rock.
*crumbles*

------------------
| Current Score: |  1
------------------
 * Armor Bonus: *  0

|#|#|#|#|#|#|#|#|#|#|
|#| |M|0|$|^|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|^|0|0|0|0|#|
|#|0|0|0|0|$|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|0|0|^|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|^|$|0|0|0|0|$|E|
|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
r
The miner digs through some ordinary rock.
*crumbles*

------------------
| Current Score: |  2
------------------
 * Armor Bonus: *  0

|#|#|#|#|#|#|#|#|#|#|
|#| | |M|$|^|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|^|0|0|0|0|#|
|#|0|0|0|0|$|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|0|0|^|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|^|$|0|0|0|0|$|E|
|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
r
The miner digs up some Hidden Treasure!
*the miner stuffs shiny gems into their bag*

------------------
```

```
| Current Score: |  5
------------------
 * Armor Bonus: *  0

|#|#|#|#|#|#|#|#|#|#|
|#|  |  |  |M|^|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|^|0|0|0|0|#|
|#|0|0|0|0|$|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|0|0|^|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|^|$|0|0|0|0|$|E|
|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
d
The miner digs through some ordinary rock.
*crumbles*

------------------
| Current Score: |  6
------------------
 * Armor Bonus: *  0

|#|#|#|#|#|#|#|#|#|#|
|#|  |  |  |  |^|0|0|0|#|
|#|0|0|0|M|0|0|0|0|#|
|#|0|0|0|^|0|0|0|0|#|
|#|0|0|0|0|$|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|0|0|^|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|^|$|0|0|0|0|$|E|
|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
d
The miner digs up a component of the Spike Armor set!
*the miner equips the spike armor*

------------------
| Current Score: |  6
------------------
 * Armor Bonus: *  1

|#|#|#|#|#|#|#|#|#|#|
|#|  |  |  |  |^|0|0|0|#|
|#|0|0|0|  |0|0|0|0|#|
|#|0|0|0|M|0|0|0|0|#|
|#|0|0|0|0|$|0|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|0|0|0|0|^|0|0|#|
|#|0|0|0|0|0|0|0|0|#|
|#|0|^|$|0|0|0|0|$|E|
```

```
|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
```

Once the miner reaches the exit, they win!

```
------------------
| Current Score: |  17
------------------
 * Armor Bonus: *  1

|#|#|#|#|#|#|#|#|#|#|
|#| | | | |^|0|0|0|#|
|#|0|0|0| |0|0|0|0|#|
|#|0|0|0| |0|0|0|0|#|
|#|0|0|0| |$|0|0|0|#|
|#|0|0|0| |0|0|0|0|#|
|#|0|0|0| |0|^|0|0|#|
|#|0|0|0| |0|0|0|0|#|
|#|0|^|$| | | | |M|E|
|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
r
The miner digs through the Exit!
Congrats! You WIN!
```

But if the miner encounters silverfish without any armor equipped, the game is over...

```
------------------
| Current Score: |  5
------------------
 * Armor Bonus: *  0

|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|
|#| | | |0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|#|
|#|0|0| | |M|0|0|0|0|0|0|0|0|0|0|0|0|0|0|#|
|#|0|0|0|$|0|0|0|0|^|0|0|0|0|0|0|0|^|$|#|
|#|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|#|
|#|0|0|0|0|0|0|0|^|0|0|0|0|0|0|0|0|0|0|#|
|#|$|0|0|0|0|0|0|0|0|0|0|0|$|0|0|0|0|0|0|#|
|#|0|0|0|0|0|0|0|0|0|0|0|0|^|0|0|0|0|0|0|#|
|#|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|E|
|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|#|

Where will the miner go?
(l)eft, (r)ight, (u)p, (d)own :
r
The miner digs into rock invested with Carnivorous Silverfish!
The miner is attacked by Carnivorous Silverfish!
AAAAAAAGHHHHHHHH!
```

And that is it. Lucky for you I did most of the game mechanics. You just need to develop the objects and add code for the virtual parts.

3 The Virtual Part

Recall that virtuals only work if you have an object of the **derived class** stored in a **pointer to the base class**. The grid in the dig_game_adventure class is a triple pointer of type entity. This is a 2D grid of entity pointers. This means that each pointer could hold any of the 8 derived objects.

A function called **destroy()** will be called from the entities in this grid. This function is virtual because it is meant to behave differently in each derived class. For example, when destroy() is called from a rock, the miner will successfully crumble it and move into the next spot. But if destroy() is called on a hardened_wall, the miner will not be able to move through it.

```
unsigned int rslt = grid[height][width]->destroy();
```

4 Important Note About "#define"

You will see these lines of code your files:

digAdventureObjects.h

```
#define NOTHING       0
#define ROCK          1
#define SILVERFISH    2
#define ARMOR         3
#define TREASURE      4
#define WALL          5
#define EXIT          6
#define MINER         7

/* empty_space and entity messages */
#define NOTH_MSG_DTSY "There is nothing here to dig..."
/* miner messages */
#define MINR_MSG_DSTY "The miner is attacked by Carnivorous Silverfish!"
#define MINR_MSG_DTOR "AAAAAAAGHHHHHHHH!"
/* hardened_wall messages */
#define WALL_MSG_DTSY "The miner attempts to dig through the hardened wall. It cannot be destroyed."
/* rock messages */
#define ROCK_MSG_DTSY "The miner digs through some ordinary rock."
#define ROCK_MSG_DTOR "*crumbles*"
/* carnivorous_silverfish messages */
#define FISH_MSG_DTSY "The miner digs into rock invested with Carnivorous Silverfish!"
#define FISH_MSG_DTOR "*the silverfish hiss as they perish to the miner's spike armor*"
/* armor messages */
#define ARMR_MSG_DTSY "The miner digs up a component of the Spike Armor set!"
#define ARMR_MSG_DTOR "*the miner equips the spike armor*"
/* hidden_treasure messages */
#define TRSR_MSG_DTSY "The miner digs up some Hidden Treasure!"
#define TRSR_MSG_DTOR "*the miner stuffs shiny gems into their bag*"
/* exit messages */
#define EXIT_MSG_DTSY "The miner digs through the Exit!"
#define EXIT_MSG_DTOR "Congrats! You WIN!"
```

digAdventureObjects.cpp

```
#define GAME_CONTINUE    0
#define GAME_EXIT_LOSE   1
#define GAME_EXIT_WIN    2
#define GAME_ERROR       -1
```

These are **named constants**. Whenever you reference one of these in your program, the compiler will directly replace the name with the value you filled

in. I might ask you to return the unsigned integer for "MINER" result, which is 7. So you can actually just type:

```
return MINER;
```

I will also ask you to print the message associated with destroying a rock. Instead of typing it out, you can just type this:

```
cout << ROCK_MSG_DSTY << endl;
```

5 Coding the Assignment

There are 5 files. I coded some partially for you.

- digAdventureObjects.h (*your code here*)
- digAdventureObjects.cpp (*your code here*)
- digAdventureGame.h
- digAdventureGame.cpp (*your code here*)
- main.cpp

I would recommend going in the order described above.

5.1 Base Class Entity

class entity

	MAS	Member	Static	Const	Type	Name	Parameters
	private	variable	N	N	char	shape	
	protected	variable	Y	N	bool	display_messages	
	protected	constructor	N/A	N/A	N/A	entity	char
	public	function	N	Y	char	getShape	
virtual	public	function	N	Y	unsigned int	destroy	
pure virtual	public	destructor	N/A	N/A	N/A	~entity	

You must add the class to **digAdventureObjects.h** and the function definitions in **digAdventureObjects.cpp**.

- **char shape -** The shape of an entity. This is what an entity will display as in the 2D grid.
- **static bool display_messages -** After the player loses or wins, there is no need to display all the messages that are printed in the destructors of all the objects. This bool tells the objects if they should print their message. Initialize this to be true.
- **entity(char) -** A constructor that needs to intialize the shape of the object using the parameter.

- **char getShape() const -** A simple getter function that just needs to return the shape.
- **unsigned int destroy() const -** This function needs to be virtual! This function will print the NOTH_MSG_DTSY message and will return NOTHING (*"NOTHING" is the named constant for 0*). This function is called if the miner attempts to destroy an empty_space.
- **~entity() -** This needs to be pure virtual! The destructor will do nothing, it just needs to exist.

5.2 Derived Class Miner

class miner : public entity

MAS	Member	Static	Const	Type	Name	Parameters
private	variable	N	N	unsigned int	spike_armor	
private	variable	N	N	int	point_total	
private	variable	N	N	bool	is_alive	
private	variable	N	N	unsigned int	height	
private	variable	N	N	unsigned int	width	
public	constructor	N/A	N/A	N/A	miner	
public	constructor	N/A	N/A	N/A	miner	unsigned int, unsigned int
public	destructor	N/A	N/A	N/A	~miner	
public	function	N	N	unsigned int	destroy	
public	function	N	N	void	givePoints	int
public	function	N	Y	int	getPoints	
public	function	N	N	void	armorUp	
public	function	N	Y	bool	isStillAlive	
public	function	N	N	void	updateLocation	unsigned int, unsigned int
public	function	N	Y	unsigned int	getHeight	
public	function	N	Y	unsigned int	getWidth	
public	function	N	Y	unsigned int	getArmor	

This class looks hefty but all implementations are very trivial. You must add the class to **digAdventureObjects.h** and the function definitions in **digAdventureObjects.cpp**.

- **unsigned int spike_armor -** The amount of armor the miner is wearing. Starts at 0. It will increase if the miner digs up armor and it will decrease if the miner is attacked by silverfish.
- **int point_total -** The amount of points the miner has. Points are gained for mining rocks and treasure. Points are lost for digging up silverfish. Starts at 0.
- **bool is_alive -** The miner could perish at the hands of silverfish. This bool tells the game if the miner survived the encounter or not. Starts as true.
- **unsigned int height -** The current height coordinate that corresponds to where the miner is in the grid. Starts at 1.

- **unsigned int width -** The current width coordinate that corresponds to where the miner is in the grid. Starts at 1.
- **miner() -** The default constructor. This inlines a call to the entity constructor and passes 'M' as the shape. All instance variables aside from the shape can be initialized as described above (0 armor, 0 points, is_alive true, height 1, width 1).
- **miner(unsigned int, unsigned int) -** The parameter constructor. This inlines a call to the entity constructor and passes 'M' as the shape. All instance variables aside from the shape can be initialized as described above, except the height and width are passed as parameters.
- **~miner() -** The destructor does 2 things. First, it checks to see if messages should be displayed (*check the entity::display_messages variable*). If true, print the MINR_MSG_DTOR message and set entity::display_messages to false. The bool is changed because after the miner dies, the game is over and no more messages need to print.
- **unsigned int destroy() -** This function is called from the miner if silverfish attack. First, print the MINR_MSG_DSTY message. Next, check to see if the miner has any armor. If there is no armor, set is_alive to false. If there is some armor, subtract 1 from the the spike_armor variable. At the end of this function, return MINER.
- **void givePoints(int) -** Function to add points to the point_total. The points to add are passed as a parameter.
- **int getPoints() const -** A simple getter function that just returns the point_total.
- **void armorUp() -** A function that adds 1 to spike_armor variable.
- **bool isStillAlive() const -** A simple getter function that returns is_alive.
- **void updateLocation(unsigned int, unsigned int) -** This function will change the miner's height and width to match the parameters. The new height is passed first and the new width second.
- **unsigned int getHeight() const -** A simple getter function to return the height.
- **unsigned int getWidth() const -** A simple getter function to return the width.
- **unsigned int getArmor() const -** A simple getter function to return the spike_armor.

5.3 All Remaining Derived Classes

class empty_space : public entity

MAS	Member	Static	Const	Type	Name	Parameters
public	constructor	N/A	N/A	N/A	empty_space	

class hardened_wall : public entity

MAS	Member	Static	Const	Type	Name	Parameters
public	constructor	N/A	N/A	N/A	hardened_wall	
public	function	N	Y	unsigned int	destroy	

class rock : public entity

MAS	Member	Static	Const	Type	Name	Parameters
public	constructor	N/A	N/A	N/A	rock	
public	destructor	N/A	N/A	N/A	~rock	
public	function	N	Y	unsigned int	destroy	

class carnivorous_silverfish : public entity

MAS	Member	Static	Const	Type	Name	Parameters
public	constructor	N/A	N/A	N/A	carnivorous_silverfish	
public	destructor	N/A	N/A	N/A	~carnivorous_silverfish	
public	function	N	Y	unsigned int	destroy	

class spike_armor : public entity

MAS	Member	Static	Const	Type	Name	Parameters
public	constructor	N/A	N/A	N/A	spike_armor	
public	destructor	N/A	N/A	N/A	~spike_armor	
public	function	N	Y	unsigned int	destroy	

class hidden_treasure : public entity

MAS	Member	Static	Const	Type	Name	Parameters
public	constructor	N/A	N/A	N/A	hidden_treasure	
public	destructor	N/A	N/A	N/A	~hidden_treasure	
public	function	N	Y	unsigned int	destroy	

class mine_exit : public entity

MAS	Member	Static	Const	Type	Name	Parameters
public	constructor	N/A	N/A	N/A	mine_exit	
public	destructor	N/A	N/A	N/A	~mine_exit	
public	function	N	Y	unsigned int	destroy	

You must add the class to **digAdventureObjects.h** and the function definitions in **digAdventureObjects.cpp**.

5.3.1 empty_space

- **empty_space() -** Inlines a call to the entity constructor. Passes a space ' ' as a shape.

5.3.2 hardened_wall

- **hardened_wall() -** Inlines a call to the entity constructor. Passes a pound sign '#' as a shape.

- **unsigned int destroy() const -** Print the WALL_MSG_DSTY message and return WALL.

5.3.3 rock

- **rock() -** Inlines a call to the entity constructor. Passes a zero '0' as a shape.
- **~rock() -** Checks to see if messages are being displayed. If true, print the ROCK_MSG_DTOR message.
- **unsigned int destroy() const -** Print the ROCK_MSG_DSTY message and return ROCK.

5.3.4 carnivorous_silverfish

- **carnivorous_silverfish() -** Inlines a call to the entity constructor. Passes a zero '0' as a shape.
- **~carnivorous_silverfish() -** Checks to see if messages are being displayed. If true, print the FISH_MSG_DTOR message.
- **unsigned int destroy() const -** Print the FISH_MSG_DSTY message and return SILVERFISH.

5.3.5 spike_armor

- **spike_armor() -** Inlines a call to the entity constructor. Passes a hat '^' as a shape.
- **~spike_armor() -** Checks to see if messages are being displayed. If true, print the ARMR_MSG_DTOR message.
- **unsigned int destroy() const -** Print the ARMR_MSG_DSTY message and return ARMOR.

5.3.6 hidden_treasure

- **hidden_treasure() -** Inlines a call to the entity constructor. Passes a dollar sign '$' as a shape.
- **~hidden_treasure() -** Checks to see if messages are being displayed. If true, print the TRSR_MSG_DTOR message.
- **unsigned int destroy() const -** Print the TRSR_MSG_DSTY message and return TREASURE.

5.3.7 mine_exit

- **mine_exit() -** Inlines a call to the entity constructor. Passes an 'E' as a shape.
- **~mine_exit() -** Checks to see if messages are being displayed. If true, print the EXIT_MSG_DTOR message and then set entity::display_messages to false. Once the miner exits the mine, there is no need to print messages anymore.
- **unsigned int destroy() const -** Print the EXIT_MSG_DSTY message and return EXIT.

That is it for these 2 files. Consider making a test main file to try out your virtuals. Try something like this:

test.cpp

```
entity *bPtr = new rock();
unsigned int rslt = bPtr->destroy();
delete bPtr;

bPtr = new carnivorous_silverfish();
rslt = bPtr->destroy();
delete bPtr;
//...
```

Once complete, hopefully you understand how this game will work. Should **destroy()** be called an object, a message prints and some result is returned. Based on this result, the miner may be able to move into the next spot. Before doing that, **delete** will be called on the space that the miner is moving into. This delete will call the object's destructor, printing a different message.

5.4 dig_adventure_game Class

class dig_adventure_game

MAS	Member	Static	Const	Type	Name	Parameters
private	variable	N	N	entity ***	game_grid	
private	variable	N	N	unsigned int	game_width	
private	variable	N	N	unsigned int	game_height	
private	variable	N	N	unsigned int	difficulty	
private	variable	N	N	miner *	player	
private	function	N	N	void	allocateGrid	
private	function	N	N	void	deallocateGrid	
private	function	N	N	entity *	allocateEntity	unsigned int
private	function	N	N	void	populateGrid	
private	function	N	N	void	moveMiner	unsigned int, unsigned int
public	constructor	N/A	N/A	N/A	dig_adventure_game	unsigned int, unsigned int, unsigned int
public	destructor	N/A	N/A	N/A	~dig_adventure_game	
public	constructor	N/A	N/A	N/A	dig_adventure_game	const dig_adventure_game&
public	operator	N	N	void	operator=	const dig_adventure_game&
public	function	N	N	unsigned int	playerMove	
public	function	N	N	void	playGame	
public	function	N	Y	void	printGrid	

I have included the skeleton for this one. Just fill out the functions that are empty. There are 5 of them:

- allocateGrid
- deallocateGrid
- allocateEntity
- moveMiner
- playerMove

Below are details on everything in this class. You will have to read through to find the description of the functions you are coding.

- **entity ***game_grid -** This is the triple pointer where the game is held. Recall that this is a 2D array of entity pointers. That means game_grid[i][j] is a **pointer**.
- **unsigned int game_height -** The height of the grid (*indexed first*).
- **unsigned int game_width -** The width of the grid (*indexed second*).
- **unsigned int difficulty -** The difficulty of the game.
- **miner *player -** A pointer to the miner object that the player is controlling. This pointer should be aliased at all times. That means that the **player pointer** will always be looking at the miner object, and a pointer within the grid will also be looking at the same object.
- **void allocateGrid() -** This function should allocate the 2D grid. First, check to see if game_grid already contains memory. If so, call deallocateGrid. Next, allocate only the first 2 dimensions. Once complete, initialize each individual pointer within the grid to nullptr. You should be left with a 2D grid of nullptrs.
- **void deallocateGrid() -** This function should deallocate the memory used for the game_grid. First, check to see if game_grid is empty. If so, you can just return. Else, deallocate all dimensions in proper order. When this function is called, it is likely the individual pointers contain objects (*rocks, the miner, walls, etc...*), so you need to check each individual pointer to see if there is an object there. If so, delete them all first. Next, you can deallocate the width and then the height.
- **entity *allocateEntity(unsigned int i) -** A helper function used to allocate different derived objects. Using the parameter passed in to return a pointer to a new object. For example, this function may be called like this:

```
1 game_grid[i][j] = allocateEntity(WALL);
```

This function call is requesting a new hardened_wall be placed in the grid at location (i,j). So, within you function, you should have something like:

```
if(parameter == WALL){
    return (new hardened_wall());
}
```

Fill out the function for all 8 cases (*WALL, NOTHING, ROCK, SILVERFISH, ...*). The empty_space object uses the NOTHING case. Also, consider returning a nullptr at the very end in case this function is called improperly.

- **void moveMiner(unsigned int, unsigned int) -** This function needs to move the miner within the grid. First, call delete for the object that miner is trying to move to. This location is passed as parameters (*new_height, new_width*). The delete is equivalent to the miner actually destroying the object and the destructor message will print. Next, assign the player pointer to the new location in the grid. Then, overwrite the miner's old location with a new empty_space object. Finally, update the miner's height and width member variables so they reflect the change.

- **unsigned int playerMove() -** This function is responsible for one turn. This function will be called many times until the miner reaches the exit or perishes to silverfish. First, repeatedly prompt the user until they enter a proper left, right, down, or up input (*I gave this to you*). Next, compute where the miner will move next (Left : -1 to width. Right : +1 to width. Up : -1 to height. Down : +1 to height.). You should not have to worry about the new coordinates being out of bounds because the miner is contained within the grid by hardened_wall objects, but you can do some error checking if you wish. Return GAME_ERROR on any cases that an error may occur.

  Once the new coordinate is computed, call destroy() on the object contained within the new location and save the result. This could be a rock, wall, treasure, armor, silverfish, empty space, or the mine exit. Now you will need a switch statement (*or some if elses*) to handle the multiple cases:

  1. **Case NOTHING :** The miner is moving into an empty space. Simply call moveMiner using the new coordinates and return GAME_CONTINUE.
  2. **Case ROCK :** The miner is mining up some regular rock. Simply call moveMiner using the new coordinates and give the player 1 point. Then return GAME_CONTINUE.
  3. **Case SILVERFISH :** The miner found some silverfish! The silverfish will attempt to destroy the miner. So, first call destroy() on the player. Also give the player -5 points. Now, check to see if the player is still alive. If they are, move the miner to the new coordinates and

return GAME_CONTINUE. If the player is no longer alive, delete the player and update both pointers that were looking at the miner to be nullptr. Then return GAME_EXIT_LOSE.

4. **Case ARMOR :** The miner found some armor! Move the miner to the new location and call armorUp. Then return GAME_CONTINUE.
5. **Case TREASURE :** The miner found some treasure! Give the miner 3 points and move them to the next location. Return GAME_CONTINUE.
6. **Case EXIT :** The miner found the exit! Move the miner to the exit and return GAME_EXIT_WIN.
7. **Case WALL :** The miner tried to dig up a hardened wall. This is not possible so just return GAME_CONTINUE without moving the miner.

The miner case is not considered here because the miner will not dig up another miner.

- **void populateGrid() -** This function was written by Mr. P. It randomly populates all the nullptrs in the grid with objects. The miner is always placed at (1,1) and the exit is always placed in the bottom right. Next the walls are put up. Then, 1 piece of armor and 1 treasure are randomly placed in each quadrant. Next, dependent on the difficulty level, 1+ silverfish will be placed in each quadrant. Finally, the remaining empty spots are filled with rocks.

- **dig_adventure_game(unsigned int, unsigned int, unsigned int) -** The constructor for a game. The height, width, and difficulty parameters are all verified so that they can work with populate grid. Then the difficulty, height, and width are set. allocateGrid is called next and then populateGrid is called last.

- **~dig_adventure_game() -** The destructor checks to see if the miner has been deallocated. Once that is handled, deallocateGrid is called.

- **void playGame() -** An infinite loop that prints the grid and then calls playerMove(). Depending on what the result of the player's move is, the game will continue or it will exit.

- **void printGrid() const -** Prints the grid with the score and armor.

- **dig_adventure_game(const dig_adventure_game &) -** The copy constructor. Will not be used but included in case you try to mess with my code :)

- **void operator=(const dig_adventure_game &) -** The assignment operator overload. Will not be used but included in case you try to mess with my code :)

6 Compiling and Executing

Compile your code with:

```
g++ -std=c++11 main.cpp digAdventureGame.cpp digAdventureObjects.cpp -g
```

Run your code with:

```
./a.out
```

Use valgrind!

```
valgrind --leak-check=full ./a.out
```

7 Submitting Your Code

Submit all 4 files to codegrade (*you don't have to submit main.cpp*).

8 Comprehension Questions

Answer these questions via the Webcampus Quiz.

1. Based on the 2 virtuals specified for the entity class, can we allocate a basic entity object? What is the name for this type of class?
2. Why is the triple pointer **entity ***game_grid** required? Why can't it just be a double pointer like the Battle Chess assignment?
3. Why do pure virtual destructors have to have a body?