Escape the Cabin: *Reflections*

Design Description

In the program the user will be playing a protagonist who is on a weekend getaway with his girlfriend. Things are not as they seem and you ultimately decide escaping from your creepy cabin before midnight is better than staying there! The game will require the user to move around different locations in the forest, collect items, and use those items to either collect more items or escape–before midnight. Each location provides the user with a different set of options and as the user collects items they gain access to other options at different locations, such as being able to siphon gas from a boat once you have an empty gas tank and a garden hose.

Each action that the user takes will take some time off of the clock (march closer to midnight), with the game ending by midnight having a different resolution than if it ends by the user getting the car up and running. The game is intended to be a bit creepy but not overly puzzlely.

Initial Design Thoughts

- Because each location will contain a set of action that can be taken, the actions that can be taken will be displayed like a series of menu options.

- I want the game to feel a bit tense, so I'm thinking that 15 minutes taken off the clock for each move is a good amount.

- When displaying the description for the car or other locations, if using the items on that location changes something significantly, the description of that location should change accordingly.

Testing Decisions

For this project, since I'm reusing systems that have proven correct in other programs, I mainly want to test that my logic on creating/destroying the map is correct and that the available options to the player is correctly changing as the circumstances of the game change. So, when the player goes to the lakeshore, unless they have the kayak, I don't want them to be able to go to the middle of the lake, but if they do have the kayak, then that menu option should be available to them.

Class Design Descriptions

Location Class (abstract): The Location class is what represents all of the different places the user can visit throughout the game. Each location has a description and holds some sort of item. Some of the optional actions afforded at a particular location are dependent on items in that the player has previously collected. The locations are connected through a series of pointers to each other in a linked structure that has no particular shape, although it's not linear. The typical getters and setters are provided and there is a method to take an item from the location and a method to print the description of the particular location.

Private data Member(s):
```
Location* north
Location* east
Location* south
Location* west
Item item
std::string description
LocationType locationType
```

Public Method(s):
```
Location( )
virtual ~Location( ) = 0
void printDescription( ) const;
void setNorth(Location* pointHere);
void setSouth(Location* pointHere);
void setEast(Location* pointHere);
void setWest(Location* pointHere);
void setDescription(std::string newDescription);
void setLocationType(LocationType newType);
Location* getNorth( ) const;
Location* getSouth( ) const;
Location* getEast( ) const;
Location* getWest( ) const;
Item getItem( ) const;
LocationType getLocationType( ) const;
Item takeItem( );
```

Player Class: The Player class represents the player of the game and provides methods for that player to interact with the game world. The player contains a pointer to their current location along with a vector of inventory items that have been picked up by the player as the game has progressed.

Private data Member(s):	Location* currentLocation;
	std::vector<Item> inventory;
Public Method(s):	Player(Location* startingLocation = nullptr);
	~Player( );
	Display a list of available actions
	bool chooseAction( );
	void printLocationDescription( ) const;
Private Method(s):	bool doAction(PlayerActions doThis);
	void printInventory( ) const;
	bool hasCan( ) const;
	bool hasHose( ) const;
	bool hasTape( ) const;
	bool hasKayak( ) const;
	bool hasGas( ) const;
	void removeItem(Item removeMe);

Input Testing Plan and Results

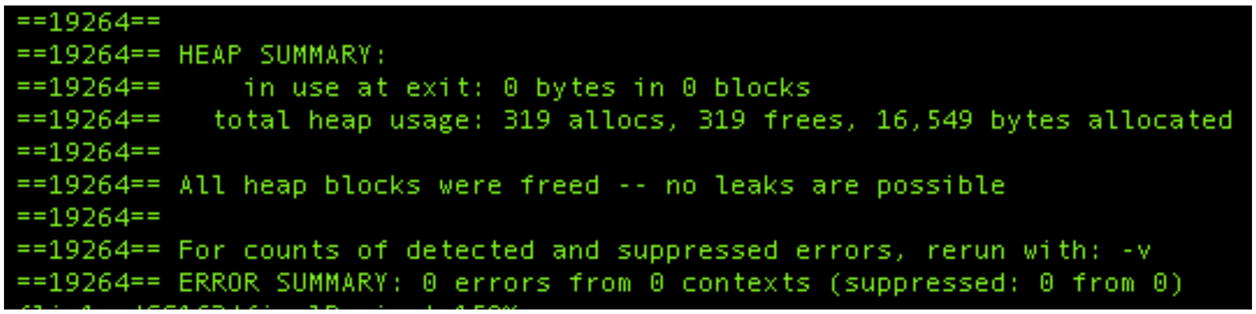
Input Testing Plan: Cabin menu

Test Case	Input Values	Driver Functions	Expected Outcome	
Boundary/Negative:	“0”	player.doAction( )	User re-prompted. 15 minutes added	User re-prompted. 15 minutes added
Boundary/Negative:	“7”	player.doAction( )	User re-prompted. 15 minutes added	User re-prompted. 15 minutes added
Positive: Within Range	“1”	player.doAction( )	User moves to forest. 15 minutes added	User moves to forest. 15 minutes added
Positive: Within Range	“6”	player.doAction( )	Shows inventory. 15 minutes added	Shows inventory. 15 minutes added
Negative: Numbers, symbols, and/or whitespace combinations	“ *4$”	player.doAction( )	User re-prompted. 15 minutes added	User re-prompted. 15 minutes added

Input Testing Plan: Lakeshore menu

Test Case	Input Values	Driver Functions	Expected Outcome	Actual Outcome
No Kayak	n/a	Player.doAction()	No option to go south.	No option to go south.
Kayak	n/a	Player.doAction()	Option to go south (onto lake) is displayed	Option to go south (onto lake) is displayed

Valgrind Memory Leak Results (No Leaks Possible)



```
==19264==
==19264== HEAP SUMMARY:
==19264==     in use at exit: 0 bytes in 0 blocks
==19264==   total heap usage: 319 allocs, 319 frees, 16,549 bytes allocated
==19264==
==19264== All heap blocks were freed -- no leaks are possible
==19264==
==19264== For counts of detected and suppressed errors, rerun with: -v
==19264== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```