Name: Program Design

Author: Nathan Johnson

Date: 06.04.2019

Description: Program design for the Final Project

PROGRAM DESCRIPTION:

The program is a single player game where the player controls a cleaning robot on an enemy ship that has been hacked. The objective is to gather materials to build a laser that will destroy the ship by causing the engines to explode. See map of the ship on the next page.

GAME FLOW

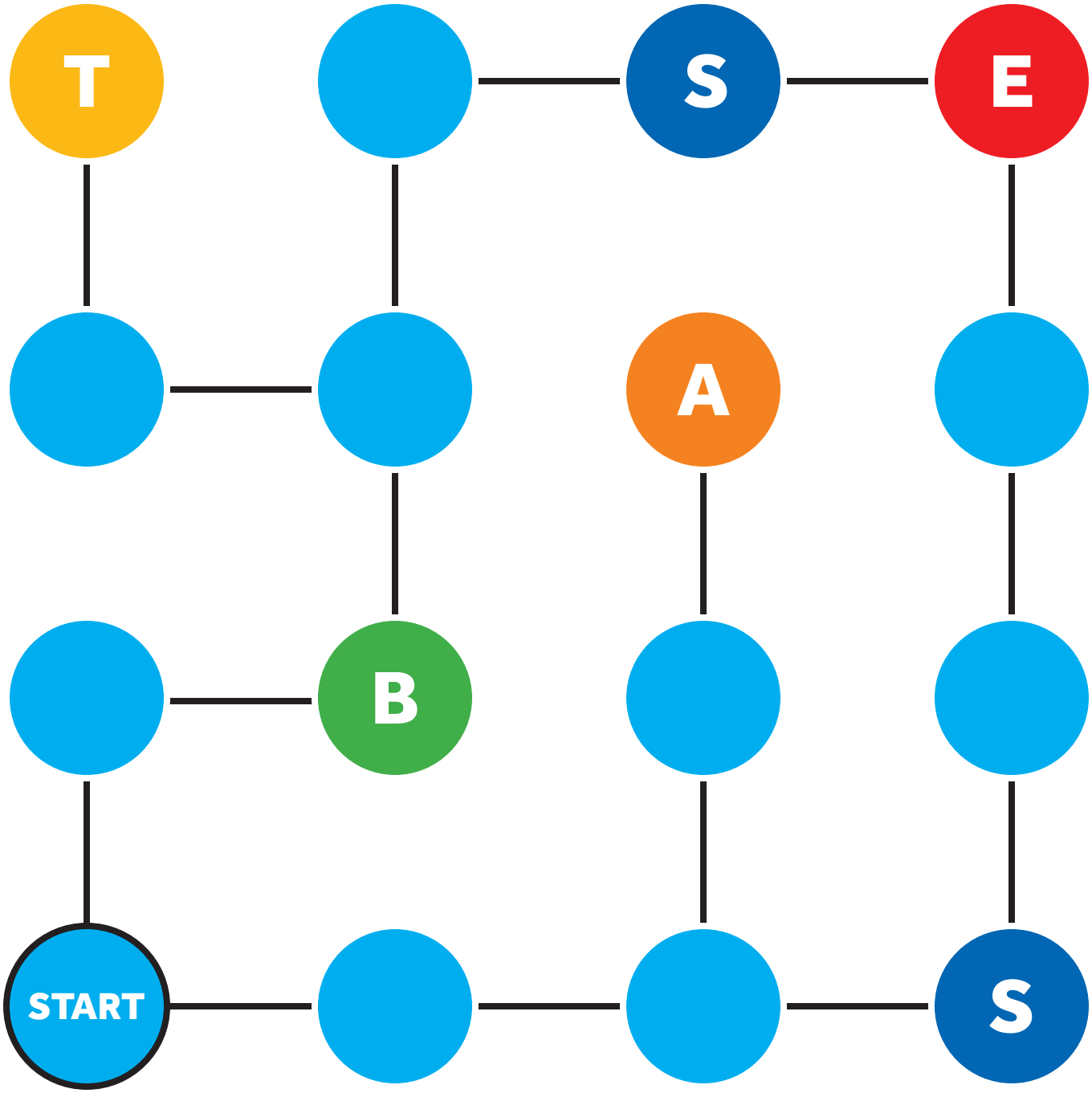
As a robot you have a limited battery, every room travelled consumes 5% of the battery, the two storage rooms have a charging station to recharge. To build the laser you need the following components:

- Arc Lamp (for an energy source, found in the armory)
- Sapphire (for the laser medium, found in the treasury)
- Metal tube (to house the laser, found in the barracks)
- 2 mirrors (for an optical resonator, found in storage rooms)

Once the components have been gathered, the player must head to the engine room to fire at the fusion reactor powering the ship's engines to cause it to explode. Room interactions are as follows:

- Standard room: choose either standard mode or stealth mode.
    - Standard mode: -5% battery, 20% chance of being detected.
    - Stealth mode: -10% battery, 5% chance of being detected.
    - If detected: -20% battery to escape
- Armory: choose either to wait for scheduled cleaning time (-15% battery) or search for the arc lamp immediately (30% chance to lose the game). Collect arc lamp unless already visited.
- Barracks: clean bob's spilled lunch (-10% battery & -1 inventory spot) or skip it (20% chance to lose the game). Collect metal tube unless already visited.
- Engine room: If you don't have the laser built: leave the room (-10% battery) or suicide charge into the reactor (90% chance to lose the game)
    If you have the laser: Fire laser at reactor (win the game) or fire laser at engine core (lose the game)
- Storage room: Collect mirror or recharge battery
- Treasury: Sneakily swipe sapphire (-20% energy) or cause a distraction by knocking down a shelf (-2 inventory spaces)

You start off with 7 inventory spaces and need 5 to complete the mission. If at any point you run out of energy, a crew member takes you to get recharged and notices you've been hacked, causing the mission to fail. You start in the lower left room room, the lines in the map are passageways between the rooms.



MAP KEY

START - Standard room
BLANK - Standard room
A - Armory
B - Barracks
E - Engine Room
S - Storage Room
T - Treasury

PROGRAM FLOW

The Space class is an abstract class representing the different types of rooms and has a constructor to show which directions link to another room. Each room has a subclass with its own event messages, items, etc. The Player class contains battery life and a struct for the inventory. The Game class creates the spaces and links for the map and has functions to implement the game with the Player and the Spaces. Main calls the menu function which lets the player exit or play a new game. The validation functions are used in various functions for input validation.

Here is a list & brief description of the files used:

main.cpp // Creates & loops the menu until the user exits

menu.hpp/.cpp // Displays menu & implements Final Project

Game.hpp/.cpp // Contains objects/functions to implement game

Player.hpp/.cpp // Containers inventory and battery life

Item.hpp/.cpp // Container for items, double-linked list

Space.hpp/.cpp // abstract class, base for different rooms

Room.hpp/.cpp // Derived from Space

Armory.hpp/.cpp // Derived from Space

Barracks.hpp/.cpp // Derived from Space

Engine.hpp/.cpp // Derived from Space

Storage.hpp/.cpp // Derived from Space

Treasury.hpp/.cpp // Derived from Space

getInt.hpp/getInt.cpp // Integer validation

main

Seed random generator

Loop Menu until user picks exit

menu() Function:

1. Play

2. Exit

Overloaded menu(string array, int) Function:

Displays the string array as a menu, the first item in the array is the title

Input's the user's menu selection and returns it

Item class – Container for items

Item Codes:

1 - Arc Lamp

2 – Sapphire

3 - Mirror (need 2)

4 - Metal Tube

Member Variables

Item* next // Pointer to next item in list

string name // Name of the item

int val // Number code for the item

Member Functions

Item(string nameIn, int valIn) // Constructor

string getName()

int getVal()

Item* getNext()

Void setNext(Item* itemIn)

Player class – Contains player inventory and battery info

Member Variables

Item* head // Start of player inventory

int itemSlots // Inventory slots remaining

int battery // Current battery life of player

int batteryMax // Max amount of battery (100)

Member Functions

Player() // Default constructor

~Player() // Destructor, frees items

void addItem(Item* itemIn) // Adds item to player inventory

    if no items in inventory

        head is item in

    else

        set item in next to the head

        set item in as new head

    remove 1 space from inventory

void printItems() // Displays items in inventory

    if no items in inventory

        display no item message

    else

        display inventory title

        while there's an item

            print the item's name

bool hasLaser() // Returns true if player has all items needed for laser

    make a flag for each component

    while there's an item

        set component flag for that item to be true

    if every component flag is true, return true

    else return false

void chargeBattery() // Sets battery to 100

void drainBattery(int drainIn) // Reduces battery life

bool slots() // returns true if inventory is full

void drainInventory(int drainIn) // Reduces max items

int getBattery() // Displays current battery life

bool batteryEmpty() // returns true if battery is empty

Space class – Base class for the rooms

Member Variables

Space* north // Pointer to the space to the north

Space* east // Pointer to the space to the east

Space* south // Pointer to the space to the south

Space* west // Pointer to the space to the west

bool visited // true if the player has been to the room before, otherwise false

Item* roomItem // Pointer to an item in the room (if any)

string name // Name of the room

bool lost // True if player loses during interaction

bool won // True if player wins during interaction

Member Functions

Space() // Constructor

vitual ~Space() // Destructor to free dynamic memory

void setDoor(int direction, Space* spaceIn) // Sets a space pointer to an existing space

1 = N

2 = E

3 = S

4 = W

bool win() // Return true if player wins during interaction

bool lose() // Return true if player loses during interaction

virtual void interact(Player* playerIn) // Controls interaction for the room, abstract function

bool getVisited() // Returns true if room has been visited, otherwise false

void print() // Displays surrounding rooms

    create a string for each direction

    For each direction (starting from north)

        If there is a pointer to a room in that direction

            Add the room's name to the direction's string

            Increment counter

    From 1 until the counter

        For each direction (starting from north)

            If there is a pointer to a room in that direction

                Add the string for that direction to the array

    pass array to menu, return users choice

Space* getNorth() // Returns the space to the north

Space* getEast() // Returns the space to the east

Space* getSouth() // Returns the space to the south

Space* getWest() // Returns the space to the west

string getName() // Returns name of room

int die(int minVal, int maxVal) // returns random number between range

Classes derived from space – Item contained and interaction shown above

<u>Game class – Creates and Plays the game</u>

Member Variables

16 spaces in the game, pointer for each space

Space* current // Pointer to the current space occupied by player

Player* robot // Pointer to the player's character

bool lost // true if player loses game

Member Functions

Game() // Constructor

    Make the map

    Make the player

~Game() // Destructor

    Delete pointers

void makeMap() // Makes the map

    for each room

        create the appropriate derived space

        create the doors to surrounding spaces as indicated in map

void play() // Implements the game

    Display intro text

    While there's battery and you haven't lost

        Take a turn

        If you win during interaction

            Win

    Lose

void win() // Display winning message

void lose() // Display losing message

void turn() // move to new space and interact with it

    move

    interact

    if you lose during interaction

        lose

    if you win during interaction

        return

    print inventory

    print battery remaining

move() // Moves player to the next Space of their choosing.

Print current adjacent rooms and get player's choice

Move to selected room

getInt() Function: Prompts for int input, returns int when successful

while valid input flag is false

Prompt for int input

Store input to string

For each char in string

Set error flag to true if non-int char

If error is not flagged

Set the input to int

Set valid input flag to true

Else

Display error message

Clear input

Return input

Overloaded **getInt(int minVal, int maxVal) Function:**

while valid input flag is false

Prompt for int input

Store input to string

For each char in string

Set error flag to true if non-int char

If error is not flagged

Convert input to int

If input is between min & max values

Store the input

Set valid input flag to true

Else

Display out of range message

Else

Display error message

Clear input

Return input

Test Case	**Input**	**Expected Output**	**Actual Output**
Run program	Run program	Program displays Menu	Program displays Menu
Select 1st menu option to start the game	1	Starts game	Starts game
Select 2nd menu option to exit	2	Exits the program	Exits the program
Select something other than either menu option	char, string, 1>#>3	Re-prompts for correct input	Re-prompts for correct input
Starting the game	None	Show intro text, start turns	Show intro text, start turns
A turn	Menu selection during interaction	Correct result of action (depending on room, see explanation earlier), display inventory and battery life	Correct result of action (depending on room, see explanation earlier), display inventory and battery life
Run out of battery	During interaction	Lose game	Lose game
Run out of inventory space	Add an item w/ no room left	Lose game	Lose game
Lose during interaction	Select interaction option w/ possibility of losing & lose	Lose game	Lose game
Successfully build laser and fire at reactor	Interaction option in engine room after collecting laser components	Win game	Win game
Game over	None	Show main menu	Show main menu

Reflection:

Do to time commitments outside of school I didn't have as much time to work on the final project as I would have liked, I wasn't able to implement a map like I had originally planned, so the player would have to look at the .pdf of the map instead of being able to see one in-game.

Other than that, the trickiest part of the program for me was designing a variable menu to choose the next room. I almost ended up printing every direction and showing a wall or something if you couldn't move there, but I decided it would look more elegant to only show possible movement options, even though it was harder to code. My solution probably wasn't the most efficient, I created several if options for each direction, with flags and counters to find the available options, build the menu and get the selection. The next tricky bit was figuring out how to know which direction that menu selection meant, because it would be different depending how many choices there were, which direction it started at, etc. Thankfully I was able to make it work after some more counters and flags, though I don't know if I could have implemented it better since I don't remember going over this kind of logic in the lectures or readings.

Deciding what the game was going to be probably took me the longest. Most of the ideas I had were way too complex for how much time I had, it was harder to try and come up with something at least slightly unique while being relatively simple and fulfilling all the requirements for the assignment. I came up with the idea I ended up going with by trying to think of a different step limit/timer than a simple counter or health, and I thought of battery life. It is similar to a health limit in practice, but hopefully at least a slight twist. I tried making my interactions be a tradeoff between hitting two hard limits, battery life, inventory space, and outright losing if you were low on both. After many run throughs of the final game, I'm relatively satisfied with the gameplay, It's challenging but not impossible to win, and the choices make a meaningful difference and there are multiple possible strategies.

I finally made an improvement to my menu function which made it much easier to use for this project. Before I had been hard coding any submenus inside the menu function, but I instead made it modular and accepting of a string array to create the menu options. It ended up working really well and I was satisfied by how easy it was to implement.

I spent some time trying to figure out how I could automate the map creation, I originally wanted to have a random map for every game with different room placements. Since I ran out of time to create the in game map, I thought it would be extremely difficult and confusing for the player, and would be too difficult for me to create playable random maps with connecting doorways with my current knowledge, I ended up just hard coding the map in the Game class.