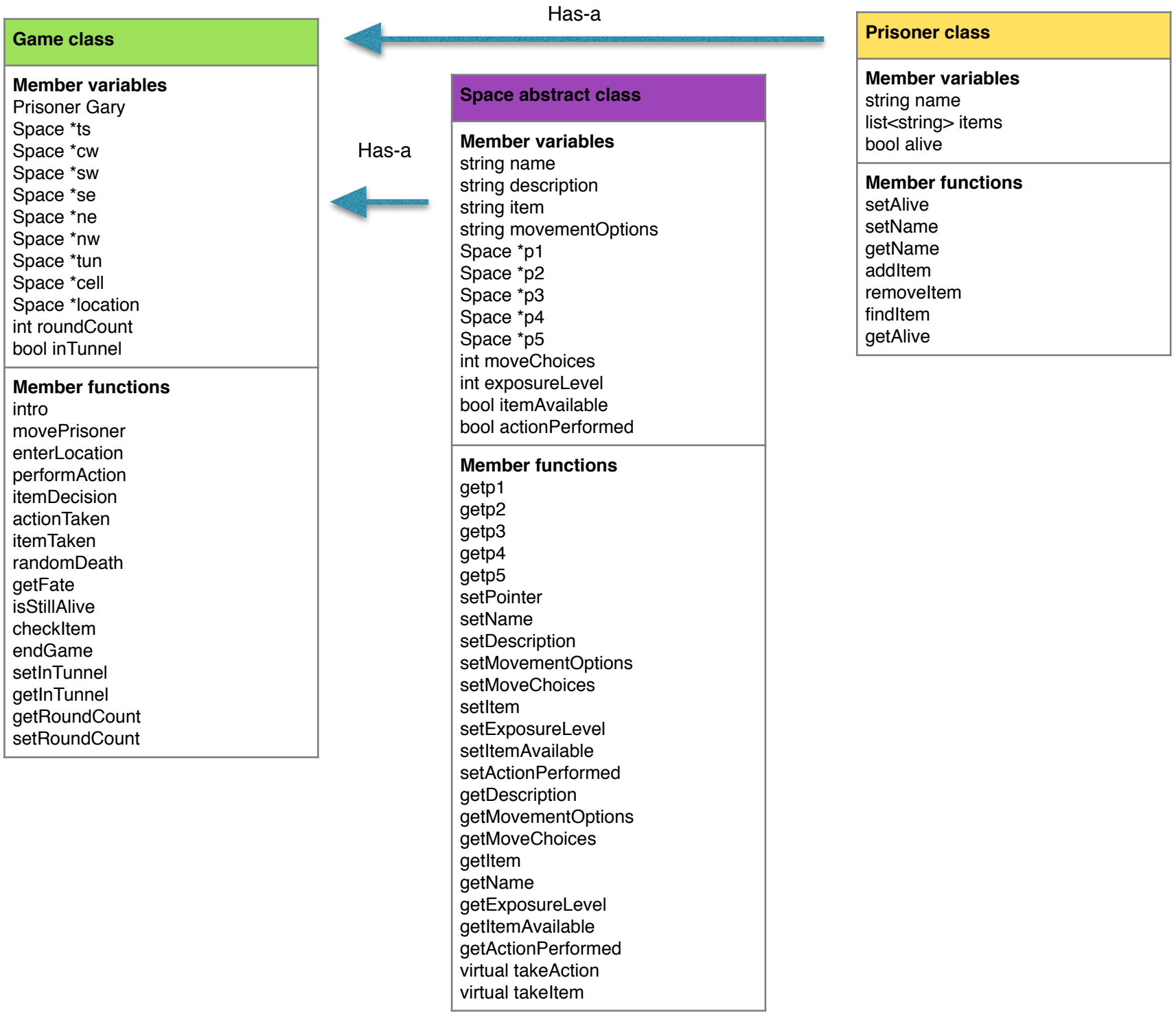
CS162 Final Project
Aaron Berns
ID #933019959

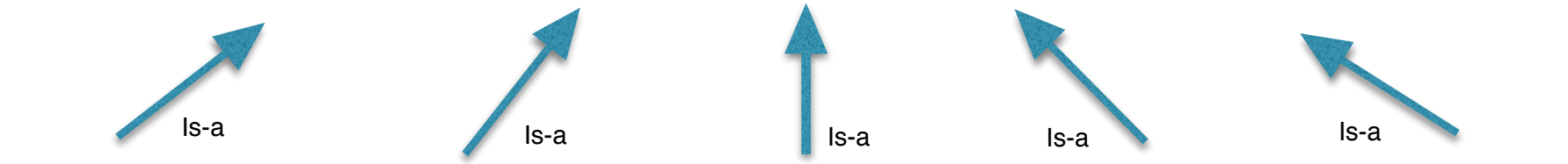
Program Requirements

This program is a choose your own adventure style text-based game. It leads the user through a series of connected spaces in a prison. The first time a space is entered, the user will be asked to change the space in some way and will be given a choice to take or leave an item. Subsequent visits will display a different short description with no interaction options. The prison, which has been taken over by the prisoners, is in the process of being taken back by law enforcement, who are shooting to kill. The user must try to make it from the starting place in the middle of the prison yard to a cell in a cell block where they will find relative safety. To gain access to the cell block, the prisoner must have the letter from a hostage and no more than one fellow inmate with him. If the prisoner has a knife, 2x4 or more than one fellow inmate, he is shot on sight even if he has the letter as well. The full version of the game may randomly kill the main character after each change in location as a way of limiting the time the user has to explore and finish the game.

For grading, the fastest way to finish the game is to choose to turn the timer off, leave the knife in the first space, move to the yard, leave the 2x4 in the second space, move to the northeast corner of the yard, take the letter, move back to the southwest corner of the yard, enter the tunnel, leave the pipe and enter the cell.

Because this is loosely based on a real event, I included an overhead map of the prison and a historical photo on which the character's situation is based. Most of the historical details came from the book *Blood in the Water: The Attica Prison Uprising* by Heather Ann Thompson, which is a good read about a troubling event.


Game class
Member variables
Prisoner Gary
Space *ts
Space *cw
Space *sw
Space *se
Space *ne
Space *nw
Space *tun
Space *cell
Space *location
int roundCount
bool inTunnel
Member functions
intro
movePrisoner
enterLocation
performAction
itemDecision
actionTaken
itemTaken
randomDeath
getFate
isStillAlive
checkItem
endGame
setInTunnel
getInTunnel
getRoundCount
setRoundCount
Has-a
Has-a
Space abstract class
Member variables
string name
string description
string item
string movementOptions
Space *p1
Space *p2
Space *p3
Space *p4
Space *p5
int moveChoices
int exposureLevel
bool itemAvailable
bool actionPerformed
Member functions
getp1
getp2
getp3
getp4
getp5
setPointer
setName
setDescription
setMovementOptions
setMoveChoices
setItem
setExposureLevel
setItemAvailable
setActionPerformed
getDescription
getMovementOptions
getMoveChoices
getItem
getName
getExposureLevel
getItemAvailable
getActionPerformed
virtual takeAction
virtual takeItem
Prisoner class
Member variables
string name
list<string> items
bool alive
Member functions
setAlive
setName
getName
addItem
removeItem
findItem
getAlive



class TimesSquare : public Space

Member variables
string desc1
string desc2
string mOptions
string actionPrompt
string actionPrompt2
string itemPrompt
string itemPrompt2

Member functions
TimeSquare()
takeItem
takeAction
changeDescription

class Catwalk : public Space

Member variables
string desc1
string desc2
string mOptions
string actionPrompt
string actionPrompt2
string itemPrompt
string itemPrompt2

Member functions
Catwalk()
takeItem
takeAction
changeDescription

class Yard : public Space

Member variables
string desc1
string desc2
string mOptions
string actionPrompt1
string actionPrompt2
string itemPrompt1
string itemPrompt2

Member functions
Yard()
takeItem
takeAction
changeDescription
setDescription2
setIP1
setIP2
setAP1
setAP2

class Tunnel : public Space

Member variables
string desc1
string mOptions
string actionPrompt1
string itemPrompt
string itemPrompt2

Member functions
Tunnel()
takeItem
takeAction

class Cell : public Space

Member variables
string desc1
string actionPrompt1
string itemPrompt1
string itemPrompt2

Member functions
Cell()
takeItem
takeAction

Class Definitions

The Game class has several private member variables. It has a Prisoner object, eight pointers to Space objects to create the eight areas in which the game takes place and one pointer to a Space object, location, that stores the location of the Prisoner object in the game. It has an int, roundCount to track how many times the character has moved and a bool, inTunnel, which indicates whether or not the Prisoner has entered the final section of the game. It has a constructor that names the Prisoner and indicates that he is alive for the time being. It creates the eight game spaces and links their pointer variables to the appropriate space. Location is set to TimesSquare, the first space. It sets the member variables for the four separate Yard objects and names all of eight of the space objects. The intro() function displays background on the main character's situation. movePrisoner() gets and validates the user's choice and uses the location pointer to update the main character's position. randomDeath() gets a random integer for the chance of death and one for choose the random death prompt of which there are three. It checks to see if the chance integer falls between 0 and the threshold for exposure for the current location and displays a deathPrompt and kills the main character if it does. enterLocation() gets the name and description for the location just entered and displays them, then updates the round count. performAction() checks to see if the current location is the first and if not prompts the user press enter, then calls the virtual takeAction() function for the current location. itemDecision() calls the virtual takeItem() function for the current location, verifies that the location is not the cell, which has a special item case, and if the user chooses to add the item to the Prisoners collection, calls his addItem() function. It sets the location's itemAvailable to false whether the user takes the item or not, giving them one chance to take it even if they come back to the space. If the location is the cell, a special decision is presented. itemTaken() and actionTaken() get and return the current location's item availability and action status. getFate() is called when the main character attempts to enter the tunnel to the cell block. It searches his list of items and based on what he has, determines if he can progress to the end of the game. The wrong items lead to death. isStillAlive() get and returns the Prisoner's life status and endGame() displays the ending message.

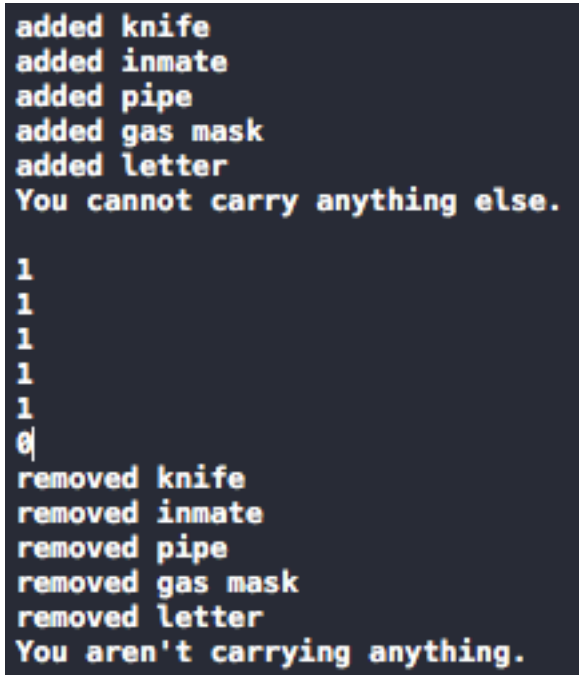
The Prisoner class has private variables for the prisoner's name, a string, life status, a bool and a list of type string to store its items. It has get and set function for life status and name as well as item related functions addItem(), removeItem() and findItem(), which add, remove and search for a specific item. The findItem() function returns true if the Prisoner has the item.

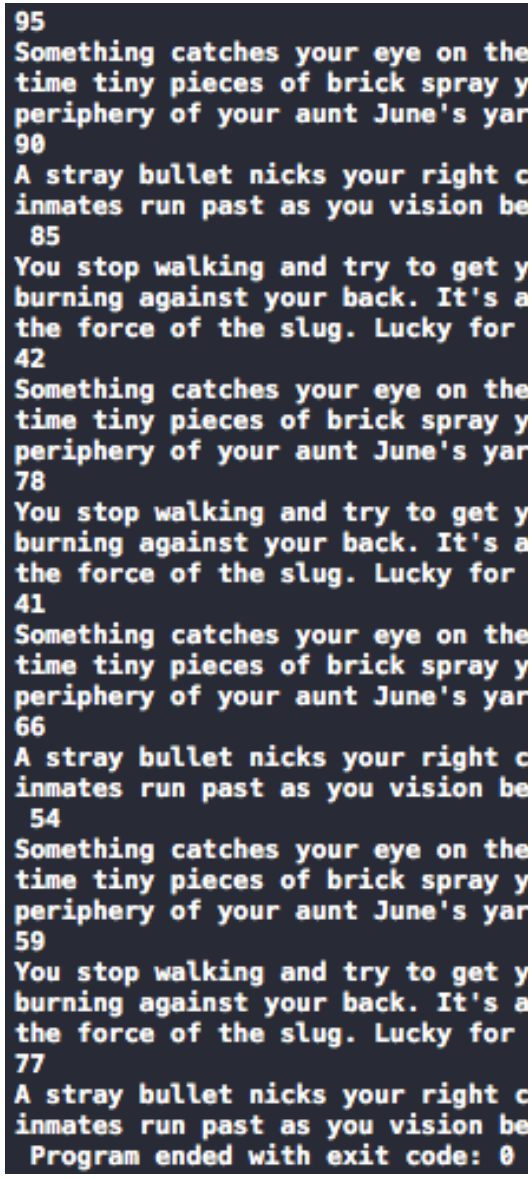
The abstract class Space has several protected variables. It has strings for the space name, description, item and movement options as well as five pointers to Space to link the derived objects to each other. It has two ints, movChoices to store each derived object's number of movement options and exposureLevel to hold the chance of random death for the space. It has two bool variables to indication the availability of the space's item and whether or not the space's action has been performed. It has set and get functions for all member variables as well as two pure virtual functions, takeAction() and takeItem(), which allow each derived class to specify the nature and mechanics of their actions and items.

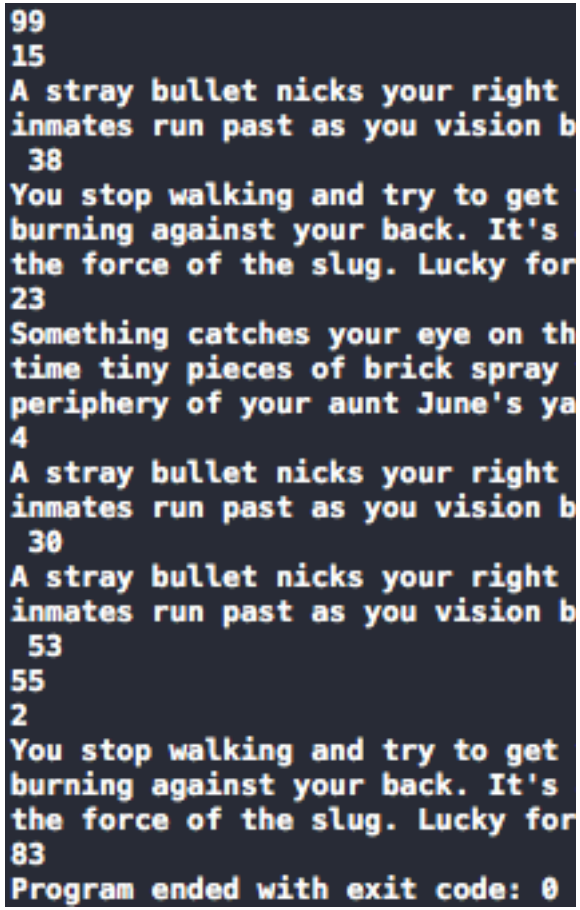
The classes derived from the Space class include TimesSquare, Catwalk, Yard, Tunnel and Cell. Each has a combination of string variables for one or two descriptions, desc1 and possibly desc2, a string that lists movement options, mOptions, one or more action prompts, actionPrompt, actionPrompt1, actionPrompt2 and one or more item prompts, itemPrompt, itemPrompt1, itemPrompt2. Each has a constructor to set it's inherited variables with its specific prompts and descriptions as well as uniquely defined takeAction() and takeItem() functions. If the class has more than one description it has a changeDescription() to update what the user is told when revisiting the space. The Yard class is instantiated four times in this game and is passed unique descriptions, prompts and options for each object and has additional set functions to individualize each Yard object.

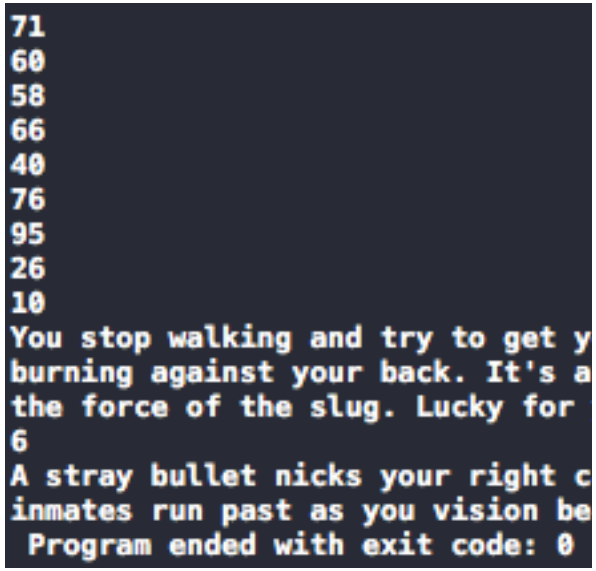
The main() function seeds srand, creates a Game object, gets and validates the timer choice and starts the game. It loops through Game's enterLocation, and movePrisoner functions until the Prisoner is killed or tries to enter the tunnel to safety. As the Prisoner attempts to enter the tunnel, Game's getFate() function is called and the Prisoner is killed or admitted based on which items he has or doesn't have. If the Prisoner gains entry, they enter the tunnel, make one more item decision and progress to the end of the game.

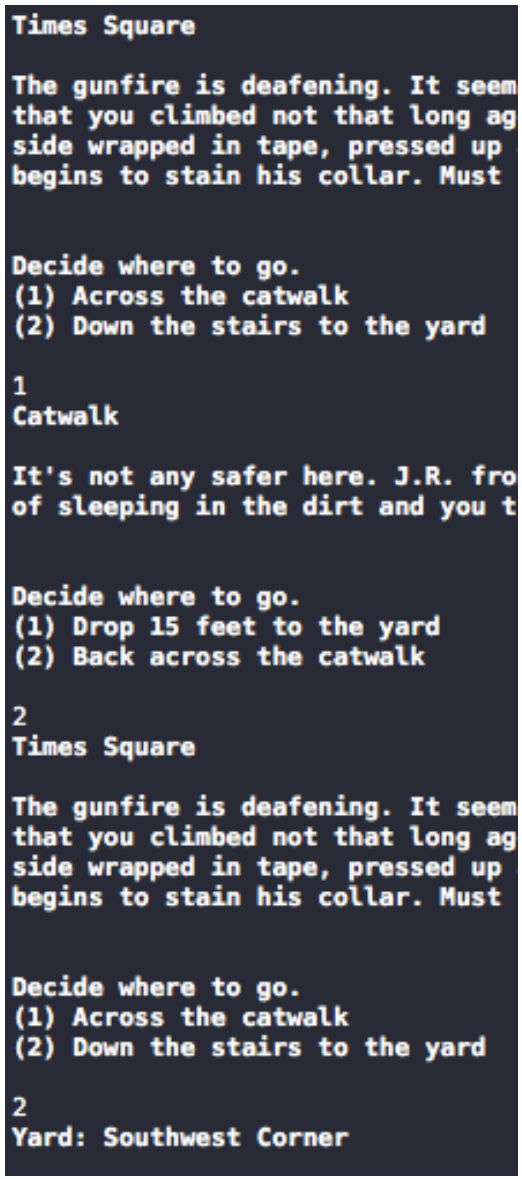
Test Plan and Results

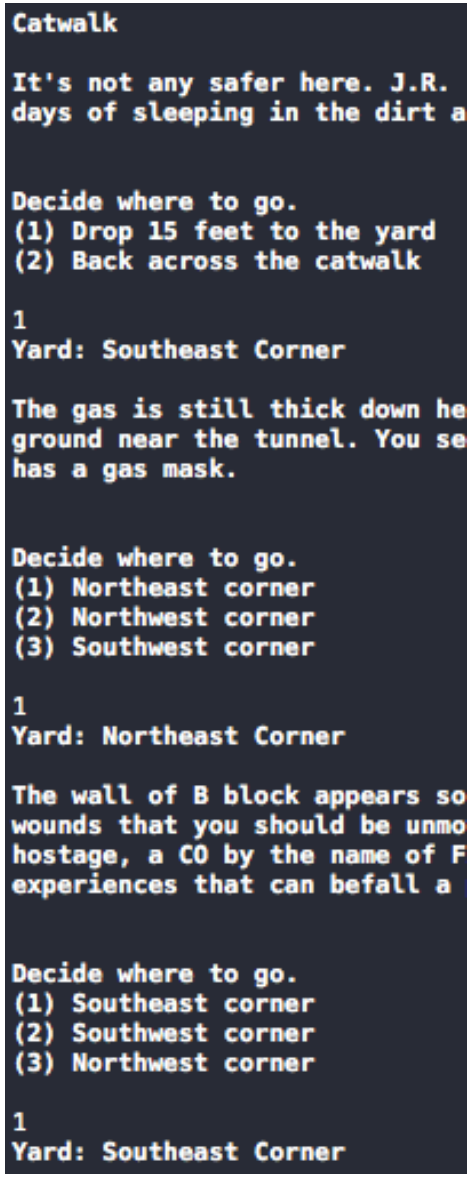
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Prisoner::addItem(), removeItem(), findItem()	add: “knife”, “inmate”, “pipe”, “gas mask”, “letter”, “sandwich” find: “knife”, “inmate”, “pipe”, “gas mask”, “letter”, “sandwich” remove: “sandwich”. “knife”, “inmate”, “pipe”, “gas mask”, “letter”, “sandwich”	main, Prisoner object, try/catch block	1st five are added, 6th is rejected, all objects stored return true for found, others false, all are removed, attempted removal of empty list sends message	

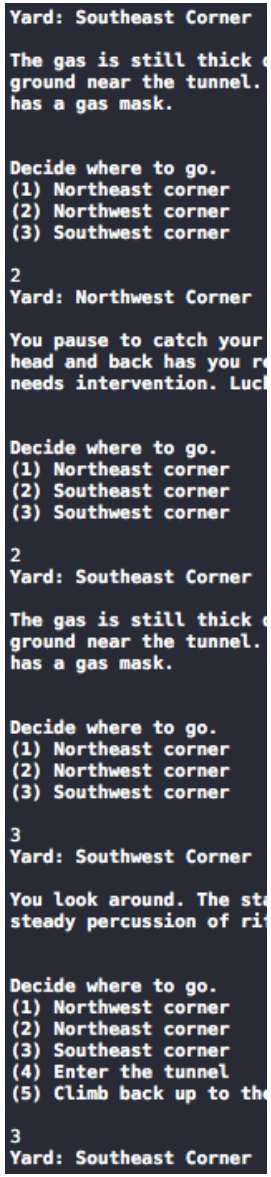
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::randomDeath	Times Square object set to exposure level of 100, function called 10 times	main, function call	random number between 0 and 100, random death (1-3) printed	

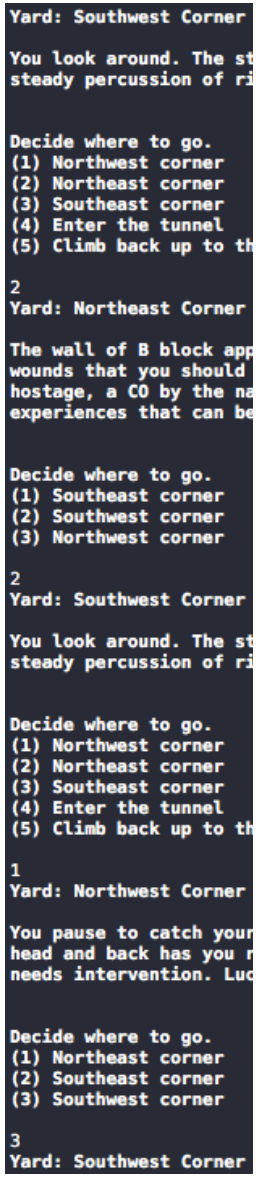
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::randomDeath	Times Square object set to exposure level of 50, function called 10 times	main, function call	random number between 0 and 100, random death (1-3) printed if number is 0-50	

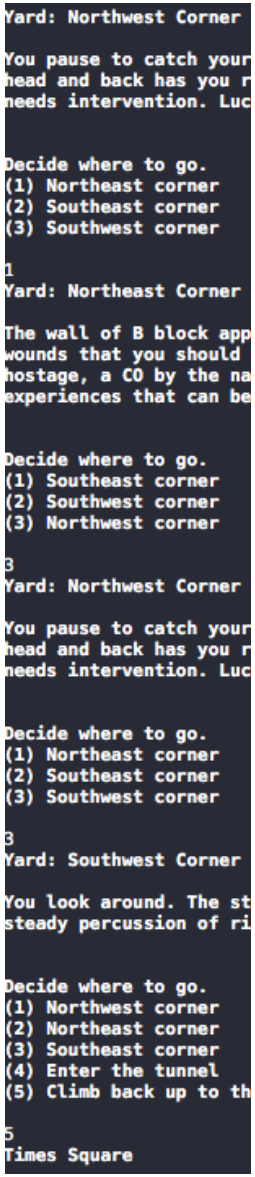
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::randomDeath	Times Square object set to exposure level of 25, function called 10 times	main, function call	random number between 0 and 100, random death (1-3) printed if number is 0-25	

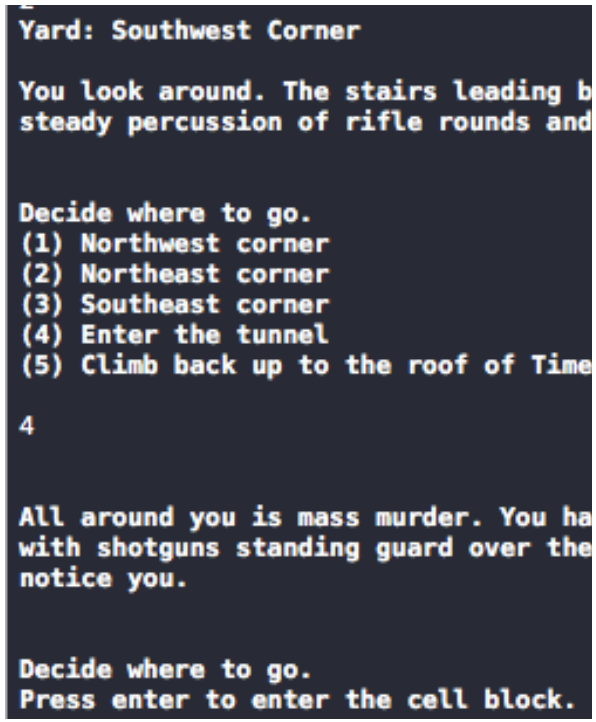
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::movePrisoner, enterLocation()	timesquare to catwalk, catwalk to timesquare timesquare to yard	main, function calls	movement matches prompt, title and description displayed upon entering	

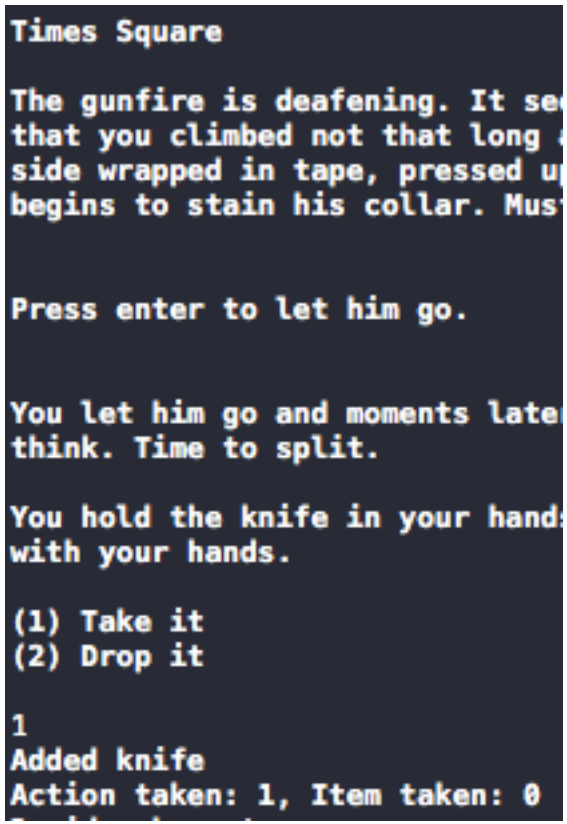
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::movePrisoner, enterLocation()	catwalk to yard, yard se to ne, ne to se, se to nw, nw to se	main, function calls	movement matches prompt, title and description displayed upon entering	

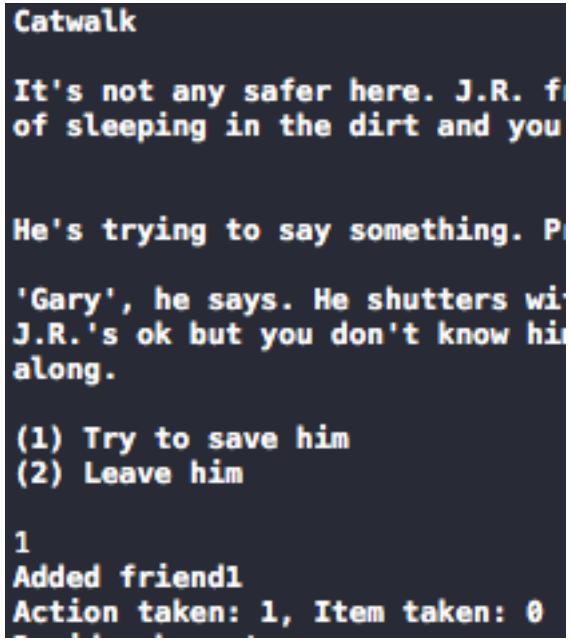
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::movePrisoner, enterLocation()	sw to ne, ne to sw, sw to nw, nw to sw	main, function calls	movement matches prompt, title and description displayed upon entering	

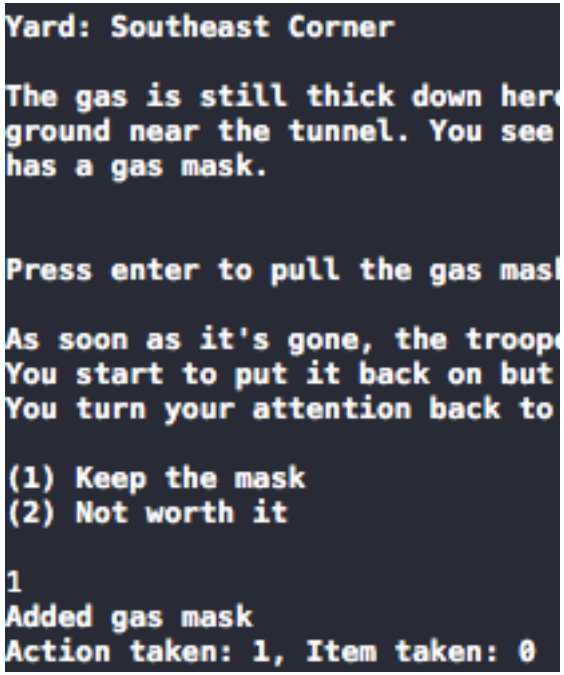
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::movePrisoner, enterLocation()	se to nw, nw to se, se to sw, sw to se	main, function calls	movement matches prompt, title and description displayed upon entering	

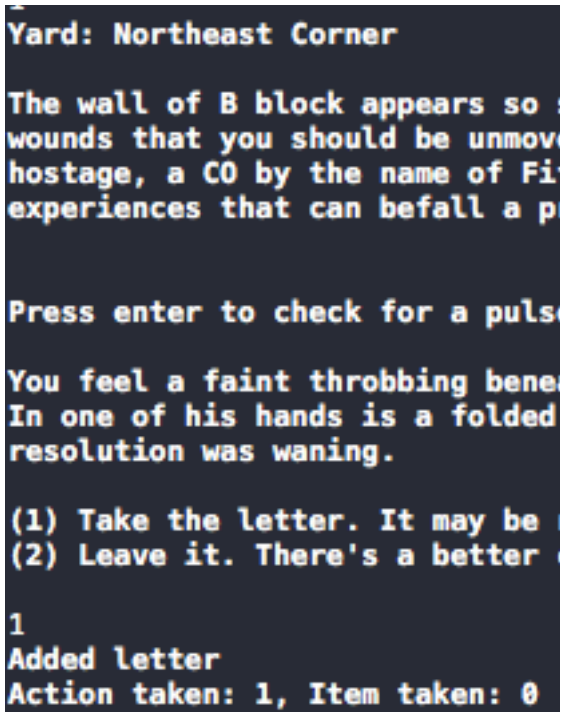
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::movePrisoner, enterLocation()	nw to ne, ne to nw, sw to times square	main, function calls	movement matches prompt, title and description displayed upon entering	

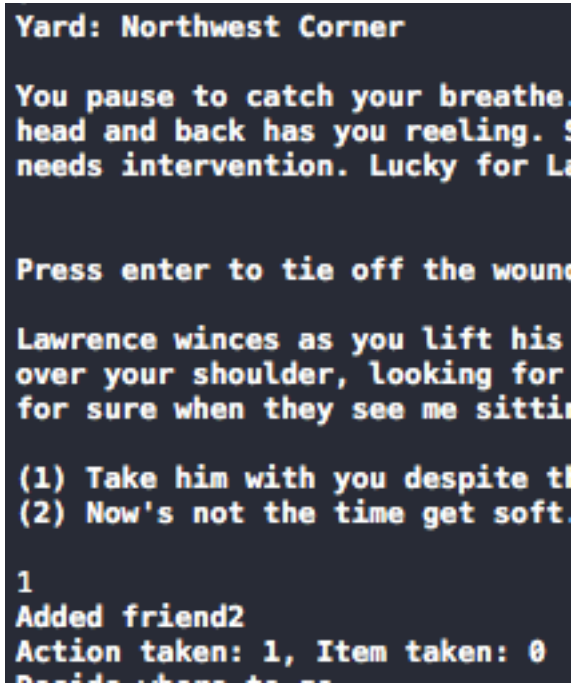
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game()::movePrisoner, enterLocation()	sw to tunnel	main, function calls	movement matches prompt, title and description displayed upon entering	

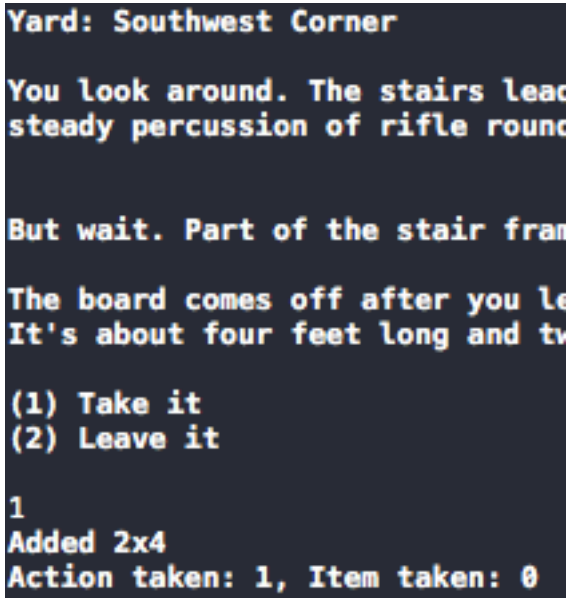
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::performAction(), itemDecision(), actionTaken(), itemTaken() Times Square	1	main, function calls	action taken 1 item taken 0 (not available)	

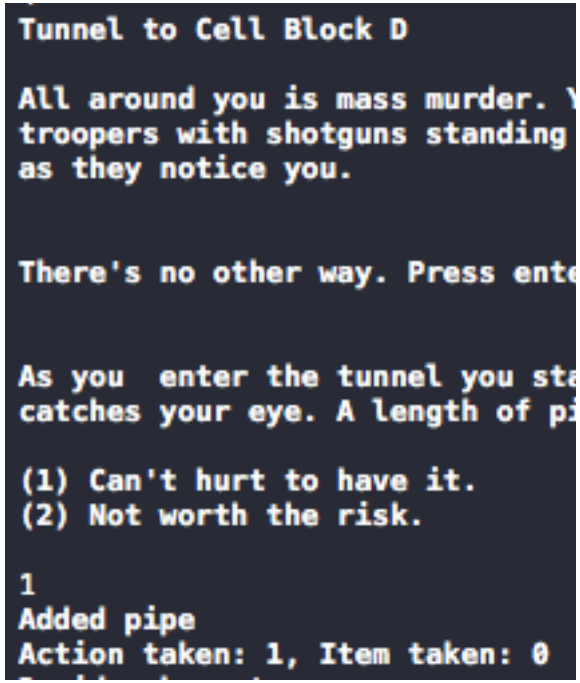
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::performAction(), itemDecision(), actionTaken(), itemTaken() Catwalk	1	main, function calls	action taken 1 item taken 0 (not available)	

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::performAction(), itemDecision(), actionTaken(), itemTaken() Yard se	1	main, function calls	action taken 1 item taken 0 (not available)	

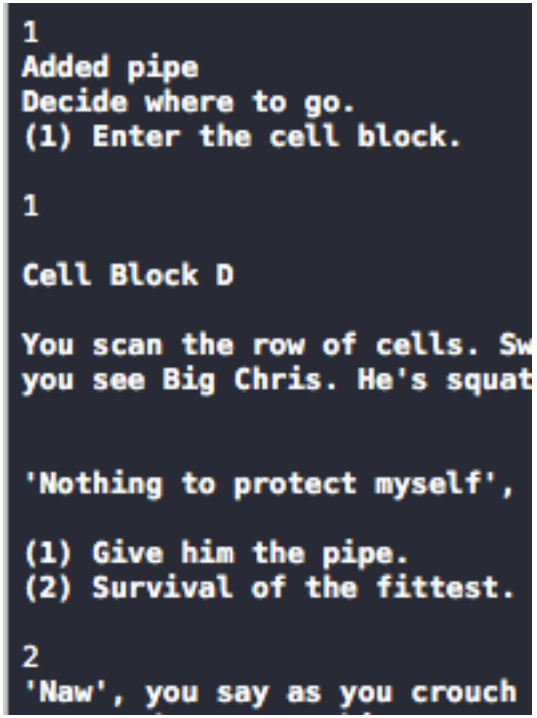
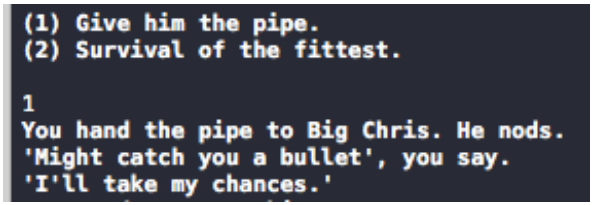
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::performAction(), itemDecision(), actionTaken(), itemTaken() Yard ne	1	main, function calls	action taken 1 item taken 0 (not available)	

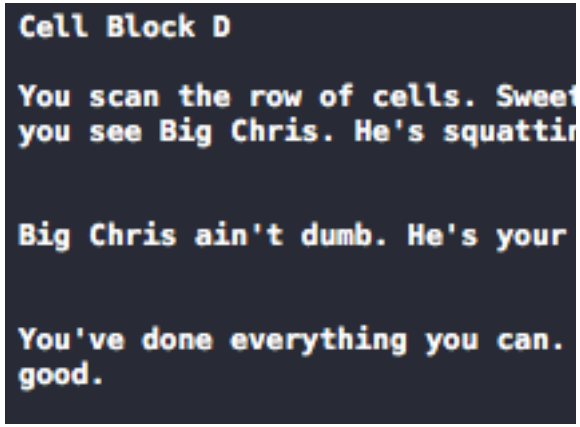
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::performAction(), itemDecision(), actionTaken(), itemTaken() Yard nw	1	main, function calls	action taken 1 item taken 0 (not available)	

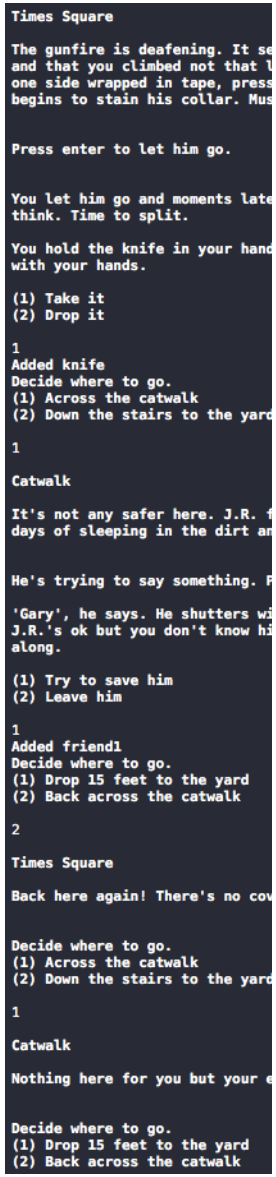
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::performAction(), itemDecision(), actionTaken(), itemTaken() Yard sw	1	main, function calls	action taken 1 item taken 0 (not available)	

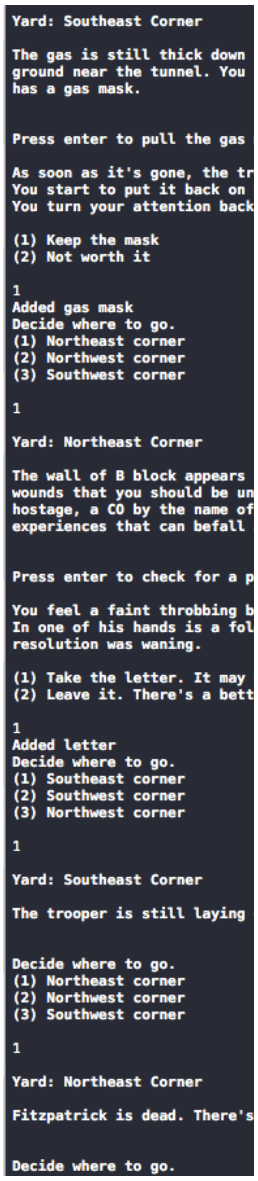
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::performAction(), itemDecision(), actionTaken(), itemTaken() tunnel	1	main, function calls	action taken 1 item taken 0 (not available)	
Game::getFate() knife death	add only knife, try to enter tunnel	Full game	Knife death message	Knife death message displayed
Game::getFate() 2x4 death	add only 2x4, try to enter tunnel	Full game	2x4 death message	2x4 death message displayed

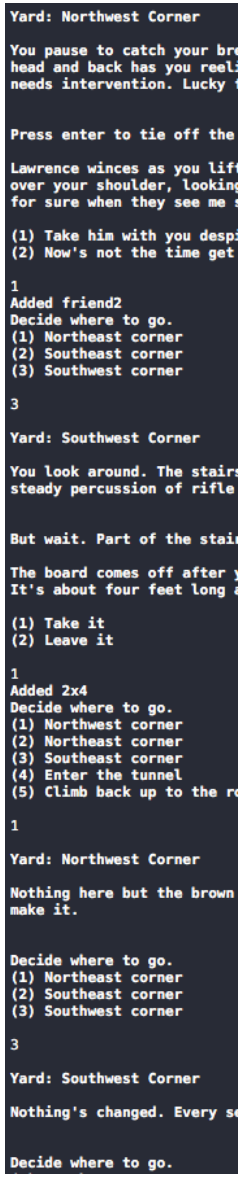
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::getFate() too many friends death	add both friends, try to enter tunnel	Full game	Too many friends death message	Too many friends death message displayed
Game::getFate() 1 friend, no letter death	add J.R., try to enter tunnel	Full game	no letter death message	No letter death message displayed
Game::getFate() 1 friend, no letter death	add Lawrence, try to enter tunnel	Full game	no letter death message	No letter death message displayed
Game::getFate() letter, no death	add only letter	Full game	passage to tunnel granted, game continues	Passage granted game continues

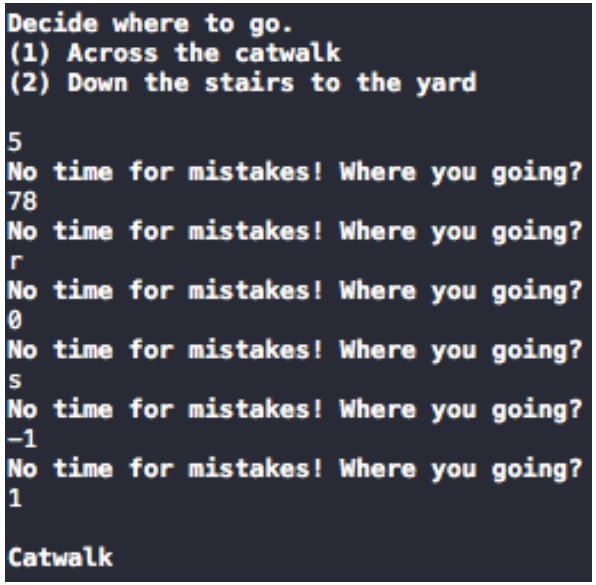
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::enterLocation() cell with pipe	pick up pipe in tunnel don't give to Chris	full game	pipe prompts choice in cell, give to Chris message	
Game::enterLocation() cell with pipe	pick up pipe in tunnel give to Chris	full game	pipe prompts choice in cell	

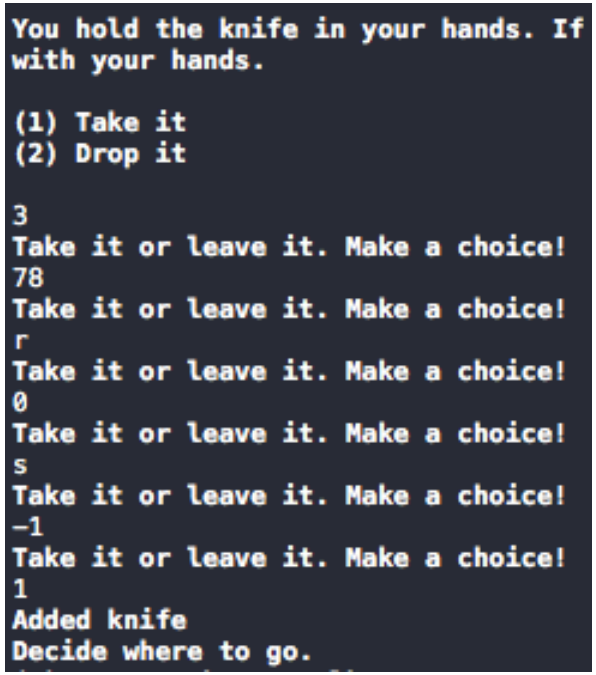
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::enterLocation() cell without pipe	leave pipe in tunnel	full game	no pipe choice in cell	

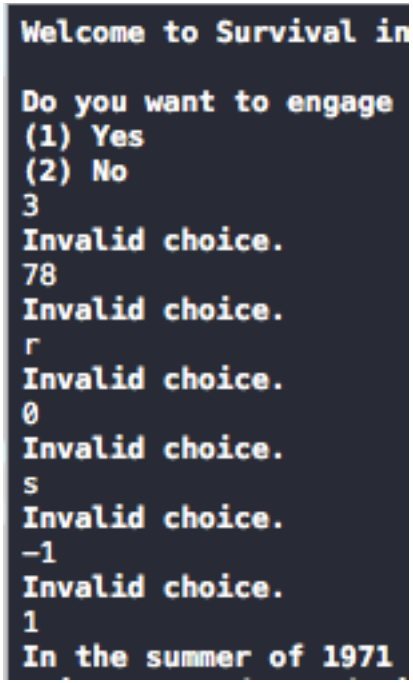
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::enterLocation() 2nd location description printed after first visit	times square to catwalk to times square to catwalk	full game	Times square desc1 action prompt and item prompt, catwalk desc1, action prompt, item prompt, times square desc2, catwalk desc2	

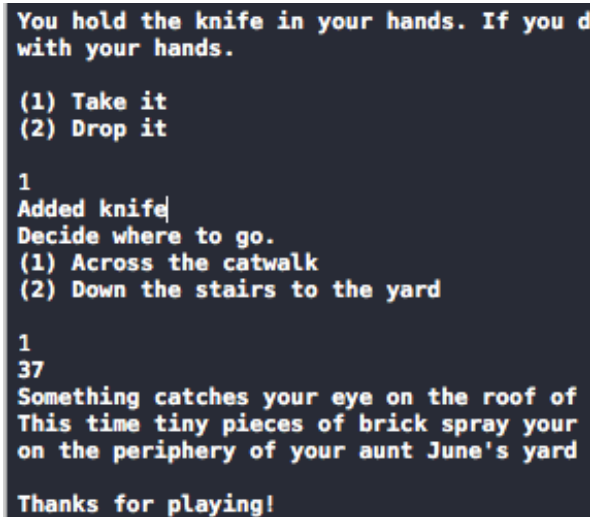
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::enterLocation() 2nd location description printed after first visit	catwalk to yard se to yard ne to yard se to yard ne	full game	Yard se desc1 action prompt and item prompt, Yard ne desc1, action prompt, item prompt, Yard se desc2, Yard ne desc2	

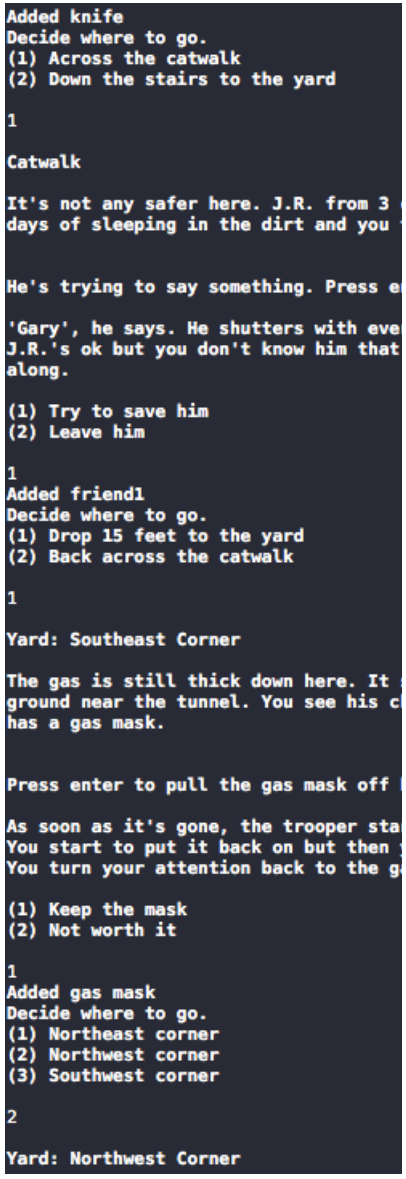
Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::enterLocation() 2nd location description printed after first visit	yard ne to yard nw to yard sw to yard nw to yard sw	full game	Yard nw desc1 action prompt and item prompt, Yard sw desc1, action prompt, item prompt, Yard nw desc2, Yard sw desc2	

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Game::movePrisoner() input validation	times square 5, 78, r, 0, s, -1, 1	Full game	error message each time until 1 is input	

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
TimesSquare::takeItem() input validation	times square 3, 78, r, 0, s, -1, 1	Full game	error message each time until 1 is input	

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
main() death choice input validation	3, 78, r, 0, s, -1, 1	Full game	error message each time until 1 is input	

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
main() random death timer on	1 at menu	full program, high exposure level set to 100	random death occurs after first move	

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
main() random death timer off	2 at menu	full program, high exposure level set to 100	no random death after several moves	

Reflections

A lot of the problem solving that took place in this project centered around inheritance. I used base-class pointers to create the derived spaces as expected and used them to call base-class public functions without a problem. I used four different object of the same derived class, Yard, to break the prison yard into quadrants, each with its own name, description, item etc. Because of this the Yard class had its own set functions and it took me a while, looking through the text and stackoverflow, to be reminded of the need to static_cast base pointers. I also struggled with whether to use set functions or the combination of derived and base constructor parameters to initialize member variables. I decided to have the unique derived objects set their own variables, inherited and otherwise, in their constructors, bypassing the base constructor. The Yard objects were initialized by the Game constructor. I searched for reasons why this approach shouldn't be used, but didn't find any. For me it made sense because of the combination of single use and multiple use of the derived objects.

Inheritance popped up again when it came time to free the memory allocated to the Space-derived objects. This proved to be challenging because the text touched on virtual destructors, but not on their role in freeing memory. Search stackoverflow eventually led to a solution and design paradigm: abstract classes must have virtual destructors and are responsible for cleaning up things that are under their control. I used the Space destructor to unlink the Space-derived objects because the derived constructors I tried to implement could not do it. The Game class constructor also couldn't unlink them without causing memory leaks, even with the setPointer function. Ultimately the combination of the Space destructor unlinking its pointers and the Game destructor freeing allocated memory led to a lack of leaks. This left nothing for the derived destructors to free or dismantle, so I left the implicit destructor in place. I ran across the best practices book on github while trying to see if this was ok and read about the Rule of Zero, which makes sense and seems to validate this choice.

The final problem I came across was the ambiguity in the assignment about using all four pointers in each space. In the description it seems like each object has to have that amount of pointers and that they we should 'try' to use them all. Then in the rubric is seems like we have to use all of them. The design of my game, which is based on an actual Prison uprising in an actual Prison, limited the movement options in some of the spaces. I gave the Space class five pointers because I have one derived space that uses all five. The rest of the spaces use three, two or one of the pointers because of the layout and design of the game. I hope that my ability to manipulate pointers is apparent even if the number of pointers each space needs fluctuates.

Overall this was a challenging and engaging assignment which forced me to clarify and solidify my understanding of several programming areas. The open ended nature has made me want to become clear on best practices given that goals can be accomplished in many ways, all of which have space and performance advantages and repercussions.