November 27 2017
CPSC 2720

Dungeon Dweller Written Report

Group Name: Kodika
Group Members: Gates Kempenaar, Reid Paulhus, Rylan Bueckert, Tomas Rigaux, Tyler Siwy

Project Description:

Our group has created a text-based dungeon exploring game. It consists of the player being able to traverse through randomized rooms, interacting with NPC characters, fighting monsters, solving puzzles, and interacting with items in order to survive as long as possible. The goal of the game is to get a high score by surviving as many rooms as possible without dying.

Planned Features not Implemented:

- Loot/Items (Keys to locked doors, better equipment (Armor or weapons), healing items to extend the player's life)
-Locked Rooms
-Fight Encounter
-Trading with a shopkeeper to purchase an item
-Inventory
    -drop/pick up items
    -interact with objects
- combat

Features Implemented:

-Displaying Rooms, displaying Characters, and Minigame interfaces.
-User-controlled movement between rooms
-Default Player System (health system, interact with NPC's)
-Inventory system (Help manage player items)
- NPCs
    -Monsters that wants to fight you
    -Puzzlers that challenge you in return for a reward
    -Have NPC interaction (Basic conversation trees)
-Randomized room generation:
    -Empty rooms
    -NPC Rooms
    -Puzzle Encounter
        -Puzzles:
            -Towers of Hanoi
            -Memory Match
            -TicTacToe
            -Connect 4
            -CodeCracker
-Shop Encounter trigger
-Fightscene trigger
-Cutscenes

November 27 2017
CPSC 2720

-High Score
-Credits/Title Card

Changes to Object Oriented Design

Character(base class):
-Health, Stamina, and Gold variables are now handled in the Character.h header
-Character.h has an ImportImg object to represent the character on the screen
-Vector of Item pointers to represent the character's inventory is implemented in the Character.h header
-Character.h now has functions to fill, modify, and return the character's inventory
-Character.h has accessor functions for health, stamina, and gold
-Character.h has modifier functions for health, stamina, and gold
-Character.h has a function to return the image representing the character and a function to draw the player to the screen

Npc(derived class):
-Has all the functions and variables inherited from character

Player(derived class):
-Name, Race, and Keys variables now handled in Player.h
-Player.h now has functions to access name, race, and keys
-Player.h now has a modifier functions for keys

RoomTree:

Node:

-Now has up, left, down, right instead of parent, left, centre, right
-Parent can be any direction, stored in variable RootDir
-currentRoom() changed to At()
-Added functions to see current height (CurrentHeight()) and to see total nodes in the tree (TotalNodes())

Item:
-NameGenerator function to create some names for Items

Weapon:
-Swords cannot be used infinitely
-Bows have a chance to miss

Consumable:
-Keys are no longer part of item
-Removed Invisibility Potion (didn't fit vision for the game)

Puzzle:
-Removed PuzzleMenu from header file since every puzzle now has a unique menu for them.

-Removed PromptUser(&screen), UserInput(&Menu), ValidCommand(); since they are all now handled by the unique menu objects instead.
-Removed ValidMove(); since it will be implemented uniquely in every puzzle.
-Added int RandomNumber(int n) const; void SecondDelay(int seconds) const; since they are helper functions that multiple puzzles call on and don't effect the class at all, left them public..
-Removed SetOptionsInMenu() since all mini-games don't use it.

Hanoi(Derived from puzzle):
-Changed the storage system for the towers from a 2d vector of ints to a vector of stacks of chars since they are designed more specifically for this application.
-Removed PromptUser(&screen), UserInput(&Menu), ValidCommand(); since they are all now handled by the HanoiMenu object instead.
-Put the screen and menu objects in the header to make the accessible everywhere within the minigame.
-Added a logic switch helper function for deciding what to do with the input.
-Added helper functions, WinCheck, EndGamePrompt, EmptyPrompt, DiscSetup, ClearTopDisc, BoardSetup.
-Removed output screen since screen has overloaded the << operator and can be made in a single function call instead.
-ValidMove became LogicSwit

ConnectFour(Derived from puzzle):
-Removed PromptUser(&screen), UserInput(&Menu), ValidCommand(); since they are all now handled by the ConnectFourMenu object instead.
-Removed output screen since screen has overloaded the << operator and can be made in a single function call instead.
-Added helper functions, WinCheck, TieGameCheck, RightDiagonalCheck, LeftDiagonalCheck, VerticalCheck, HorizontalCheck, EndGamePrompt, EmptyPrompt, BoardSetup, IsInputOutOfScope, ResetGame
-Renamed AiMove to playAi
-Added MovePiece function
-Removed output screen since screen has overloaded the << operator and can be made in a single function call instead.

TicTacToe(Derived from puzzle):
-Removed PromptUser(&screen), UserInput(&Menu), ValidCommand(); since they are all now handled by the TicTacToe object instead.
-Removed output screen since screen has overloaded the << operator and can be made in a single function call instead.
-Added helper functions, WinCheck, TieGameCheck, EndGamePrompt, EmptyPrompt, BoardSetup, SetCurrentPlayerChar, IsInputValid, IsIntInputValid, SetCurrentPlayersChar, ResetGame, RightDiagonalCheck, LeftDiagonalCheck, VerticalCheck, HorizontalCheck, ConvertCharCoordinateToIndex

November 27 2017
CPSC 2720

-Added MovePiece function

MemoryMatch(Derived from puzzle):
-Removed PromptUser(&screen), UserInput(&Menu), ValidCommand(); since they are all now handled by the ConnectFourMenu object instead.
-Removed output screen since screen has overloaded the << operator and can be made in a single function call instead.
-Added Helper funtions: Peek, PeekAtBoard, SaveBoardToScreen, RandomNumber, ConvertCharToIndex, RandomlyInsertIntoTable, IsInputvalid, SetInputs, CheckInput,IsInUsedPairs,IsCharAlreadyMatched, IsOdd, SecondDelay.
-Went from two vectors to four, making an easy way of checking pairs, randomizing the table each game, and insuring that no symbols are incorrectly put into the table more than once

CodeCracker(Derived from puzzle):
-Removed PromptUser(&screen), UserInput(&Menu), ValidCommand(); since they are all now handled by the CodeCracker object instead.
-Removed output screen since screen has overloaded the << operator and can be made in a single function call instead.
-Added helper functions: EndGamePrompt, ValidAnswer, IsRiddleUsed, InitialPrompt, SetRiddleInMenu, MakeRiddleUsed, UnusedRandomRiddle, DeathCheck, ImportRiddles

Room:
-Created ‘Points’ to locate exits in rooms
-Include a character now to use for cutscenes
-Created randomizer for room type
-Created helper function to find the points in a room
-Created helper function to see if room is ‘complete’
-Created helper functions to return private variables
-Created helper functions to align in each direction
-Created helper functions to draw images to the screen

Cutscene:
-Created an entirely new class that uses a background image and the animated image to simulate a cutscene when switching between rooms
-Created cutscenes for intro and outro and monster events

Image:
-Created helper functions to return private variables
-Overloaded every ‘Align’ function to be able to align to either a screen or another image, and each with the ability to include values or not

ImportImg:
-Created helper functions to return the file string

November 27 2017
CPSC 2720

-Created copy constructor and assignment operator overloads

DefaultImg:
-Created copy constructor and assignment operator overloads

Screen:
-Created overloads for the Fill(), DrawBorder()
-Created function to print multiple screens side by side

SlotScreen:
-Deleted Class altogether - was not needed

GameState:
-Added Player pointer parameter
-Added functions that return: Player, Menu, Screen, RoomTree, and Current State.

ExploreState (Derived from GameState):
-Added copy constructors and assignment operator (big 3 as well as one extra copy constructor for GameState to ExploreState).
-Added Helper functions RunInput and SetState to help add functionality to menu options and swapping between states.

InventoryState (Derived from GameState):
-Added this state for to help view the player's inventory.

DialogueState (Derived from GameState):
-This was never finished or implemented
-Not deleted, but not utilized.

Menu:
-Added member function AddOption to help with creating options for the menu.

MinigameMenu (Derived from GameMenu):
-Added helper function SetQuery so that the statement in the Menu can be changed depending on which minigame/puzzle is using a Minigame derived menu class.

ConnectFourMenu (Derived from MinigameMenu):
-Added function to return the column chosen by the player to place their piece.

TicTacToeMenu (Derived from MinigameMenu):
-Added struct Coord to set coordinates x and y, x being a character from A-C and y being an integer from 1-3.

November 27 2017
CPSC 2720

-Added function GetCoordinates to return the Coord struct so that the coordinates can be read for the TicTacToe game.

MemoryMatchMenu (Derived from MinigameMenu):
-Added struct Coord to set coordinates x and y, x being a character from A-D and y being an integer from 1-4.
-Added function GetCoordinates to return the Coord struct so that the coordinates can be read for the MemoryMatch game.

RiddleMenu (Derived from MinigameMenu):
-Added helper function GetInput to return an integer for the answers to the CodeCracker games.

Kodika C++ Coding Conventions 2017

Formatting
Operators are to be separated by spaces
Ex.
cout<<"string"<<1+2<<"string<<endl; (Not okay)
cout << "string" << 1 + 2 << "string" << endl; (Acceptable)

Line Length
If feasible, each line of code should be at most 80 characters long.

Spaces vs. Tabs
Use only tabs for indentation.

Loops, If's, and Switch Statements
-Braces are optional for single-statement loops, and if statements.
-Switch statements can have curly braces or not, depending on your preference.

Punctuation, Spelling and Grammar
-Comments should be as readable as narrative text, with proper capitalization and punctuation.

Curly Brackets
-Curly brackets should always start on the line below a function declaration, loop, or if statement (Unless its a 1 line loop or if):

```
for(int i=0 i<size; i++)
{
//do stuff
}
```

November 27 2017
CPSC 2720

General Naming Rules
-Names should be descriptive; avoid abbreviation.

Type/Variable/Function Names
-General names (**Functions, classes, typdef**) start with a capital letter and have a capital letter for each new word, with no underscores: MyExcitingClass, MyExcitingEnum.
-Variable names use standard camel notation: int myExcitingVariable.

Constant Names
Variables declared constexpr or const, and whose value is fixed for the duration of the program, are named in all caps. For example const int VALUE=0;

File Extensions
Headers: .h
Implementation: .cpp

Comment Convention
-Every function header and definition should have a brief description of what the function does, including input parameters and their descriptions
-Comments should be descriptive, use sentences, avoid abbreviation
-File descriptions should be in the form:
//
///memory_match.cpp
///\author
///CPSC 2720-Howard Cheng-Dungeon Dweller
//

Error Handling Strategies Used

Minigames:
-Since Minigames were very modular they handled all of their own error handling independently of the main game.
-Invalid input was detected using the customized Menu objects for each minigame and then handled in the main. Since the player was not permitted to leave the games before they completed them, a loop was used, re-prompting for output.
-An error is thrown if the player pointer is null when passed into the minigames, since this would be a fatal error for the game there is no catch in the main and it simply ends the game.
-An error is thrown in the codecracker if the text file containing the riddles is not opened properly. Since this would also be a fatal error in the game, this error is not caught, and simply ends the main game.

RoomTree:
-RoomTree throws exceptions if it gets invalid inputs. Because these errors can only get triggered internally, if they are thrown, something is broken and the game prints an error message and quits

Item:
-If the NameGenerator function fails, it throws a runtime_error that is caught in the item constructors, then gives the item a generic name
Menu:
-HandleInput checks the input from the user to make sure it's a single character and that that single character matches the associated the integer value of that character in the options map.

Debugging and Optimization Issues

Cutscenes:
-Cutscenes required that the screen be reset and reprinted many times a second, while updating the locations of images, and checking for bounds. An issue that caused lag was using the command: system("clear"). By reducing the usleep() time each 'frame', I was able to prevent screen tearing and maintain a clear picture for the most part.
Room:
-Room generation takes time and is most noticeable when generating upwards to 100+ rooms in a loop. This is not an issue however since a single room is generated at a time while traversing. Rooms themselves had generation problems due to image placement and centering, as well as dealing with both even and odd image width values.
Puzzles:
-Could break up some of the longer functions into smaller helper functions to increase readability and modularity.

Non-Automated Testing

-Minigames
-Created a small test main to do unit testing on each minigame before implementing it into the main.

Hanoi:
-Checked all kinds of invalid inputs (Strings, chars, digits) trying to crash the game
-Tried to perform invalid moves (moving a larger disc onto a smaller one) using every command option.
-Had each group member play the game checking to make sure the win checker is always successful.

TicTacToe:
-Checked all kinds of invalid inputs (Strings, chars, digits) trying to crash the game.
-Checked different combinations of inputs of the correct type ((char, int), (int, char)) making sure no errors occurred.
-Tried all combination of coordinates to make sure pieces would be placed correctly.
-Tried x and y coordinate sizes larger and smaller than the board size, making sure that no values would be drawn outside of the bounds of the game.

-Forced the computer player to win in all 4 ways (Left diagonal, Right Diagonal, Vertical, Horizontal).
-Played the computer after a reset (As a result of a player loss) to make sure that the game would still function properly after a reset.
-Beat the computer in all 4 ways(Left diagonal, Right Diagonal, Vertical, Horizontal).
-Tied the computer then successfully played another round after a reset.

ConnectFour:
-Checked all kinds of invalid inputs (Strings, chars, digits) trying to crash the game.
-Tried all combination of columns to make sure pieces would be placed correctly.
-Tried column sizes larger and smaller than the board size, making sure that no values would be drawn outside of the bounds of the game.
-Forced the computer player to win in all 4 ways (Left diagonal, Right Diagonal, Vertical, Horizontal).
-Played the computer after a reset (As a result of a player loss) to make sure that the game would still function properly after a reset.
-Beat the computer in all 4 ways(Left diagonal, Right Diagonal, Vertical, Horizontal).
-Tied the computer then successfully played another round after a reset.

MemoryMatch:
-Checked all kinds of invalid inputs (Strings, chars, digits) trying to crash the game
-Checked different combinations of inputs of the correct type ((char, int), (int, char)) making sure no errors occurred.
-Tried all combination of coordinates to make sure pieces would be placed flipped and placed correctly if a match was found.
-Tried x and y coordinate sizes larger and smaller than the board size, making sure that no values would be drawn outside of the bounds of the game or incorrect squares would be flipped..
-Had each group member test the game to make sure that the board would be randomly generated every time and that the win check was functioning correctly.

CodeCracker:
-Checked all kinds of invalid inputs (Strings, chars, digits) trying to crash the game.
-Answered questions wrong until the player lost enough health for a game end to be triggered.
-Answered different combinations of wrong-right questions ensuring that the win checker was always successful.
-Had each group member try the game to ensure that the riddles were being selected randomly from the list with the correct associated answers each time.

-Screen
-I would try to place images in different places, both less than or greater to the bounds on the screen ensuring it printed properly. I allowed functionality to draw only a portion of the image to the screen, so testing ensured that that was possible and that it would not break.

-Room

-I would generate different rooms ensuring that they included everything based on the type of room. I also would generate every variation of each room as well as generate rooms randomly.
-During development, I had to test the random generation to ensure that the correct images were placed and in the correct positions. These would be checked by displaying the x,y values of each image.
- Images that have variations (pillars) had to have a check to ensure multiple 'pillars' did not populate the same room, and this was tested extensively.

-Image
-Testing involved ensuring that the effects called on each image were applied
correctly, these included Fill(), Empty(), etc
-Ensured that all of the overloaded functions responded correctly
-Tested extensively to ensure the copy constructors worked correctly

Lessons Learned

-Learned about the issues encountered with class abstraction such as:
    -Multiple Inheritance.
    -Passing around pointers of different types of objects and how they react with inheritance.
    -How modularity affects compatibility and overall design of the project.
-Became familiar with the state design pattern.
-Became much more familiar with debugging tools such as valgrind.
-Using git for version control.
-Configuring doxygen READMEs.
-Using scripts for automated testing.

-Coding independently and then bringing modules together is extremely challenging without constant communication. Groups should work together, in person, for a more seamless project production.
-Miscommunication is prevalent when things are described over text or through voice chat. It is much easier to work together on a project if you meet up multiple times per week and work together in person.
-Always give yourself extra time to complete a task, it will inevitably take longer than expected.
-Be open to criticism, constructive or otherwise, someone might have an idea that is better than yours, nobody knows the best way to do everything.
-Similarly, try not to be rude about suggesting alternatives to other group members.
-Designing a plan to ensure all pieces are implemented smoothly is extremely important, this was noticed late into the project when last minute changes were difficult due to design limitations.

What We Could Have Done Differently

-Connecting classes earlier rather than later to assess bugs early on.
-Schedule coding sessions twice a week where the group gets together and works in person on the project. Fixing bugs as they come up instead of working around them and trying to fix them later on.
-Schedule a completion date two weeks prior to the actual due date, using the last two weeks for debugging and optimization.

-Ensure planning is done to an even higher extent
-Discuss and plan a solution to a larger issue rather than working around it temporarily, thus reducing work in the long run.

Known Defects

-Screen flickers during cutscenes due to print speed
-Memory leaks in game state, and character.
-Had to hard code character into the game because of issues with the characters classes.
-TicTacToe menu causes a segmentation fault upon deconstruction when leaving the minigame