ComPuzzle Specification Documentation

Pennsylvania Western University of California

Senior Project I: Software Engineering

(CMSC-4900-001)

With Dr. Chen

Team Members:

Abigail Dehart

Brianna Dulik

Brandon Mastin

Cameron McGill

Instructor Comments/Evaluation

Table of Contents

Abstract\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5

Description of the Document\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

Purpose and Use\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

Intended Audience\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

System Description\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

Overview\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

Environment and Constraints\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

End User Profile\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

User Interaction\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

Hardware Constraints\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

Software Constraints\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

Time Constraints\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

Cost Constraints\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

Other Concerns\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

Acceptance Test Criteria\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11

Testers\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11

Criteria for User Acceptance\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_12

Integration of Separate Parts and Installation\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13

System Modeling\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13

Functional\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13

Entity Class Diagrams\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15

Class Name/Description/Type\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15

Dynamic: Statechart\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_17

Dataflow Sequence\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19

Components/Tools Needed\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19

Resource List\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_21

Appendix: Glossary\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_21

Appendix: Team Details\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_22

Appendix: Workflow Authentication\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_23

Appendix: Writing Center Report\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_24

**Abstract**

Our project is ComPuzzle, a 2D computer puzzle game. Our goal is to create a simple and fun video game that will teach the user a fundamental understanding of basic programming concepts. This game will include a character that the user will manipulate with the arrangement of supplied blocks of pseudocode. We will create this platform on the Godot gaming engine using Godot’s built in coding language. This documentation will provide a more detailed specification on the steps we will take to create this gaming platform — Including everything that the completed product will encompass in its finality.

**Description of Document**

Purpose and Use

This document will include an overview of the steps and methods we will use to complete our project, ComPuzzle. This will give an explanation on how we will make this gaming platform, how we plan on testing our game, the general constraints we will encounter along the way, criteria we plan to meet, the user’s interaction with our game, etc. This will not be all-encompassing, but a detailed summary of the specifications our project will meet. This document will become a binding contract when the terms and statements contained within are accepted by the development team.

Intended Audience   
 This document’s intended audience includes the development team and the client/user. Being a binding contract, the client and development team both must hold the team accountable for these goals presented. As such, the steps included will prove helpful for the team to look back on to ensure we stay on track with the criteria we laid out. The client’s purpose in reading this writing is also to ensure that all criteria will be met in the final product. It is of high importance that all criteria mentioned be met in this binding contract.

**System Description**

Overview

ComPuzzle will be a fun gaming application used to aid in the of learning basic coding problem-solving skills. There will be given blocks of pseudocode on one side of the screen, and a character that can be manipulated by said pseudocode on the other. This character must navigate through a maze full of obstacles. When the user arranges this pseudocode in a working order, the character will be able to move and overcome obstacles. The arrangement of these pseudocode blocks will be the only way to manipulate the character. In the chance that the user incorrectly arranges pseudocode, they may not pass an obstacle. If an obstacle is not passed, the user may have to restart the level if the character was put in peril. They may alternatively be given leniency to retry until correct. This leniency will depend on the hardness of the level. After all obstacles are overcome, the level is completed. The user may then move on to the next, harder level.

Environment and Constraints

End User Profile:

The target audience will be anyone interested in learning problem-solving skills related to computer programming. This could include someone wanting to play a fun game to make them think, or a computer science student looking to sharpen their critical thinking skills in their field. Someone with no coding experience at all would be able to learn to play our game. We want this to be an entry-level way to begin code-related problem-solving skills. With progression leading to harder levels, higher levels of coding knowledge and problem-solving skills will be needed. Someone with more coding experience may pass through levels faster than a user with none until a hardness level more fitting to them is reached. Our game is for anyone interested in coding.

User Interaction

The user will be required to use a mouse and keyboard for the game. The mouse will be the primary interface the user will use. The user will use the mouse to interact with the game menu, the interface, and the controls of the game. The user will interact with the keyboard for simple things like entering a name.

Hardware Constraints

The hardware requirements for this project would be a working keyboard, mouse, monitor, and computer. The keyboard will be used for asking for a save file name or player name. The mouse will be the main instrument used in the playing of the game. The mouse will be used for the drag and drop function of the code pieces and to interact with on-screen buttons. The monitor will be used for the output display of the program to allow the user to see. The only last piece that would be required for the user to have would be the computer itself, it would be very difficult to play a computer game without a computer. The computer itself would need connection to the internet, so a router and either cable or Wi-Fi card would be needed. The game should not be very processor intensive so any processor made after 2015 should be fine.

Software Constraints

On the software side, the user would need to have a windows installation. We will be using windows 11 for the program, but it should be able to be played on older versions. We chose windows because it is the widest spread operating system for consumers. The game will be played on people’s personal computers, so making it available to the most amount of people is the wisest. We will be using the game engine Godot for the development of the game. We chose Godot because it is a good game engine for a 2d clicking game, but most game engines should be able to produce this as it is not a very complex design. Godot is a free option for us because we are not making this for commercial use. This game engine also has a large community with experienced users who are willing to help other users. Godot uses its own programming language called GDScript as well as C#, which is a popular programming language that offers free assets for early development or general use. The Godot engine is primarily for windows machines. This is just another reason for us to use Godot.

Time Constraints

As this is an academic project, our time constraints are limited to the length of one semester—15 weeks. It will be important for each group member to allocate time every week to work on this project. With various responsibilities including other classes and personal, out-of-school matters, organizing time slots in our schedules to work on this project is important. Having leaders of each work segment is greatly important in the completion of this project timewise. Every respective leader will allocate time out of their personal schedule to complete work while also assigning work/deadline goals to the other group members. We will then hold each other accountable for these goals, work quality, and given deadline. To avoid procrastination, we will begin this method of workflow as soon as the semester begins. This will allow for our project to be completed in an organized and timely manner.

Cost Constraints

For the game’s development, the resources we will be using are free. Since there is no hardware other than the device running the game, there is no additional cost for components. If this game was meant for commercial use, there would likely be additional costs; this could include a license to the game engine (Godot is a free open-source option), buying assets, and finding a distributor for the game. Other cost concerns may include software to create our own assets and images, such as photoshop. Audio aspects of our game will be free as we plan to create music/sound effects ourselves on free recording software which we will then import into our game.

Other Concerns

Other concerns could include reliability. Reliability focuses on ensuring the game is bug-free and works as intended. Potential defects slipping under our noses is a possibility, and we will do all we can to avoid this. Detrimental, game-crashing error slips are possible and could ruin the user’s experience. We need to make sure that our completed product is as polished and error-free as possible. Our game must function as the dependable, bug-free game we propose it to be. We need to keep the user in mind when creating our game; we must include as much testing as possible with a wide variety of people with different gaming/coding backgrounds to keep our game bug-free and playable. Reliability is a must, and we must keep up to our standards.

Acceptance Test Criteria

Testers

Testers for our game will vary from our team to people who are unfamiliar with programming. We will start by testing the core functions of the game ourselves, seeing what works and what does not. We will then progress onto people who are less and less programming savvy. We first want people who know how the game is supposed to work to test it. They may find a bug that people who do not know how to program might miss, such as logic errors or actions being taken out of order. After that, we want people who are unfamiliar with the core concepts of computer programming to test our game. They are more likely to do things you are not supposed to do or that do not make any logical sense. This could be something as simple as spinning around in a circle until an integer overflow happens and crashes the game. Even after the game is finished, the people who play our game may find other errors. They will then become testers as well. Using a wide variety of testers will help find the greatest number of errors.

Criteria for User Acceptance

1. The user should be able to install, run, and quit a game.
2. Player can save and load progress
3. The robot will follow the code snippets
4. The player can arrange the code blocks
5. The game can determine if the robot reaches the goal
6. The robot cannot walk into a wall
7. Levels get more difficult

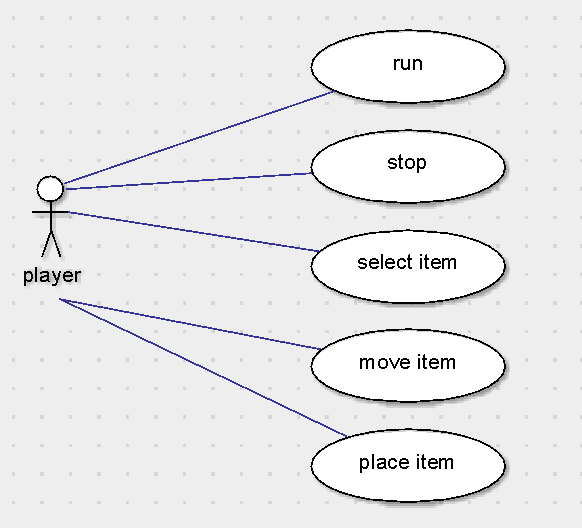
Integration of Separate Parts and Installation

Installation will be easy since it is only a program. The user will be able to download the program online which they will have to extract since it will be a zip file. After they do that, there will be an executable file that they have to double click to run. It is as simple as that.

**System Modeling**

Functional: Use Cases & Scenarios

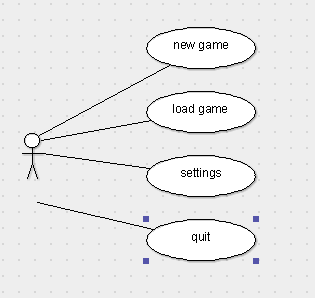
Figure 1:



Use Case Figure 1 Description:

This use case scenario depicts the player’s gameplay inside of a level. The “run” button will run and perform the pseudocode that the user arranged. “Stop” will stop running these actions. “Select item” is the depiction of selecting a block of pseudocode by left clicking with the mouse. “Move item” is meant to describe moving this block, or item, of pseudocode into place. This will be done by holding in the left mouse button and dragging. “Place item” will happen when the user is no longer holding in the left mouse button and the pseudocode will be placed.

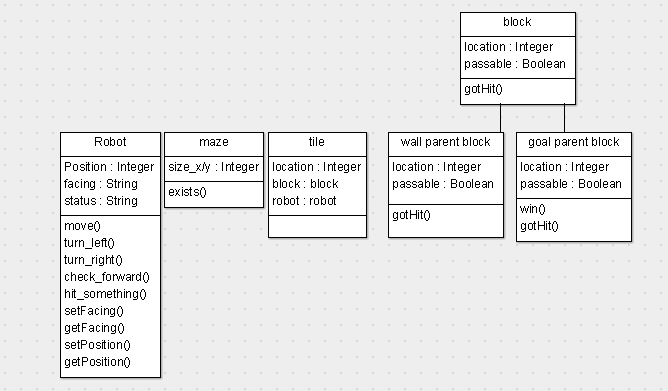
Figure 2:



Use Case Figure 2 Description:

This use case figure depicts the players options after opening the game. Options will include “new game”, which will prompt the user to type a character name that will be inputted by keyboard. After a name is chosen, the user will be brought to level select. Alternatively, if “new game” is not selected, the player may load a previous saved game instead. This will load whatever chosen character name and progress was made in the chosen save file. The “settings” option will include UI, display, and controller options for the user to customize if preferred. These first three options will all include a “back” button which will lead you back to the previous page. Then, there is the “quit” option which will exit and close the program.

Entity: Class Diagram



Class Descriptions

This diagram shows different classes and functions we will need to program to create our game.

The robot class

* Position: The robot needs a position on the grid, so we know where it is and where it needs to go with each step.
* Facing: We need to know which direction the robot is facing so that when the robot walks forward, we know the direction of north, south, east, or west.
* Status: We also need a status to know if the robot is currently doing an action or if it is idle.
* The move and turn methods will visibly change the robot, either its direction or its location.
* Check forward will look one tile in front of the robot and return a string if there is an object in front of it; such as a wall, goal, or any other object that may be introduced.
* Hit something will just check to make sure the robot is not colliding into an object, so no invalid moves are made

Maze class

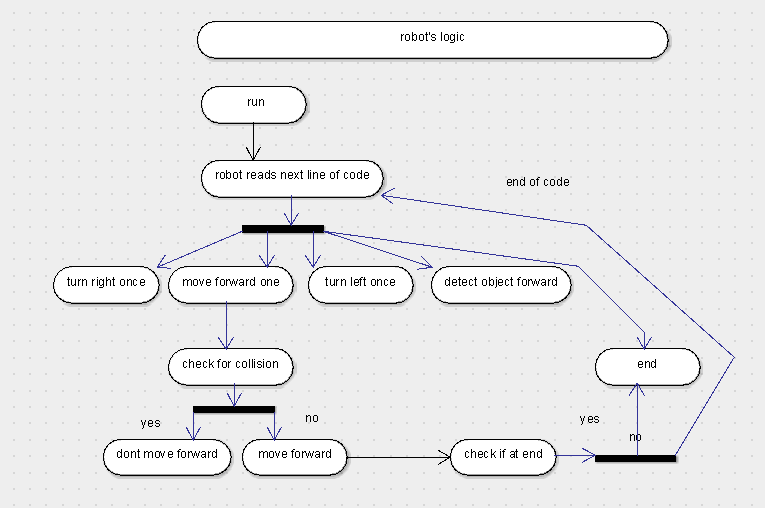
* The maze will have a size in both x and y coordinates. The maze may have dynamic shapes, other than rectangle, but still use an x/y coordinate grid.
* The only method we have for it is just to make sure the map exists.

Wall class

* The wall needs a location, x/y. There will be many walls. Walls will be required to surround the maze so that the robot stays in the grid. There will be other obstacles that are placed in the maze, but they will be using the same wall class.
* Walls may or may not be passable, so there needs to be a check for that.
* The walls need to know if they had been interacted with, because they may interact with the robot if they get hit.

Goal class

* The goal needs a location to place it on the maze
* The goal only has one method to show the victory screen when a player reaches it.

Dynamic: Statechart

States

* Idle
* Moving
* Turing left / right
* Hitting object
* Arrived at goal

Transitions

* From idle
  + Move forward
  + Turn left
  + Turn right
* From moving
  + Collision detected
  + Goal reached
  + Nothing happened, go idle
* From turning
  + Idle

Dataflow Sequence Diagram

A screenshot of a computer program

Description automatically generated

This diagram displays the user’s interaction with our game. The user will start the game and select a level. Once the level loads, the player can begin moving the pseudocode blocks into order. Once the run button is clicked, the robot will follow whatever arrangement the user submitted. Once the move is completed, a check will be performed to see if the user has completed the maze or not. If not, more pseudocode arrangements must be submitted until level completion. This is how the data will flow in gameplay.

**Components/Tools Needed**

As we are making a video game, not many tools are needed to bring our ideas into fruition. Here is a list of everything we will need:

* Computer/laptop- We cannot create a video game without a computer. Without a computer/laptop our video game could not exist.
* Display- Of course, we will need some sort of display not only for the graphics of our game, but to be able to code at all.
* Keyboard- A keyboard will be needed for typing our code and testing certain aspects of our game.
* Mouse- A mouse will be helpful during coding. More importantly, it will be necessary during the testing of our game. Testing the functionality of the clicking, dragging, and dropping of our pseudocode blocks in-game will require a mouse.
* Internet- The Internet will be needed for multiple reasons. We need the internet for our online group communication and sharing of coding progress. It will also be needed to download our game. It will be needed if any outside resources are used, such as downloading an audio software device.
* Windows- Windows is a widely used operating system that we will program our game on. We are choosing windows 11, the newest version. We will try to ensure compatibility with some older versions of windows, as well.
* Godot- Godot is the biggest piece of our needed tools. Godot is an open-source game engine that is perfect for coding 2D games. We will be using Godot’s built in language, although C# is also supported. This engine includes a built-in debugger which will help greatly in testing and debugging.
* Processor- As long as our computers/laptops include a processor that is older than 2015, we should be able run/test our game just fine.

**Resources**

*Godotengine/godot*. (2020, September 29). GitHub. <https://github.com/godotengine/godot>

*Introduction — Godot Engine latest documentation*. (2024). Readthedocs.io. <https://godot-doc.readthedocs.io/en/3.0/about/introduction.html>

**Appendix: Glossary of Terms**

Godot: Open-source game engine, used to program games.

UI: User interface, includes how the user and computer system interact.

Pseudocode: Step-by-step, informal description of a computer algorithm.

Class: In computer programming, is something made to create objects and provide reusability. Defines attributes and methods(functions) that the objects created from it will have.

Bugs: Includes any type of error or mistake in code that could cause a crash/unexpected output.

Processor: Central processing unit, allows programs and applications to work.

Hardware: Physical components of computer, such as CPU, RAM, motherboard, etc.

Software: Instructions, programs, and procedures running inside of the computer.

**Appendix: Team Details**

Work was completed as a team effort. All ideas and concepts were talked about before and during writing. Any confusions were met with a group conversation to ensure that we are all on the same page. The workflow leader was Cameron McGill. Different writing sections were designated to each group member to complete by the workflow leader. Every person worked on their respective writing section. After completion of each section, the rest of the group would proofread/edit/revise when necessary. This paper was a collaboration between the whole ComPuzzle team.

Abigail Dehart- Abstract, Purpose and Use, Intended Audience, Overview, End User Profile, User Interaction, Other Concerns, and Appendix: Team Details.

Brianna Dulik- Testers, Criteria for User Acceptance, Integration of Separate Parts and Installation, and Reference List.

Brandon Mastin- Functional, Class Name Description Type, Attributes, States, Events, Transitions, and Appendix: Workflow Authentication.

Cameron McGill- Hardware Constraints, Software Constraints, Time Constraints, Cost Constraints, Dataflow Diagrams, Components/Tools Needed, Entity Class Diagrams, and Appendix: Glossary

**Appendix: Workflow Authentication**

I, Abigail Dehart, acknowledge and accept the specifications presented in this writing, including the agreements and plans mentioned.

**** 11/18/2024

I, Brianna Dulik, acknowledge and accept the specifications presented in this writing, including the agreements and plans mentioned.

**** 11/18/2024

I, Brandon Mastin, acknowledge and accept the specifications presented in this writing, including the agreements and plans mentioned.

**** 11/18/2024

I, Cameron McGill, acknowledge and accept the specifications presented in this writing, including the agreements and plans mentioned.

**A black and white logo

Description automatically generated** 11/18/2024

**Appendix: Writing Center Report**

Writing center report via email on 11/19/2024 completed by Caroline Krofcheck.