Software Requirements Specification (SRS)
Puzzler

Team: **Puzzler (Group 9)**
Authors: **Chris Peters, Max Fitzgerald, Shkamb Tafarshiku, Elijah Miles, Ebenezer Oloyede**
Customer: **Massachusetts Department of Education**
Instructor: **James Daly**

1 Introduction

This Software Requirements Specification (SRS) document provides a detailed description of the product, including its purpose, scope, overall system behavior, constraints, and specific functional and non-functional requirements. It also defines the system's context, and major use cases. The document is intended to guide development and validation.

1.1 Purpose

The purpose of this SRS is to clearly define the functional and non-functional requirements of Puzzler, an educational puzzle game for middle-school students, so that developers, and customers share a common understanding of the system's behavior. This document serves as the basis of agreement between the development team and the customer.

1.2 Scope

Puzzler is an educational software application designed for students in grades 7-8. It's purpose is to support learning in three subject areas: mathematics, science, and english in accordance with Massachusetts Department of Education standards detailed later. This is accomplished via structured interactive puzzle-based activities. The application belongs to the educational technology and game based learning domains.

Puzzler aims to reinforce core academic concepts by providing age-appropriate puzzles that require students to apply skills such as arithmetic, vocabulary development, reading comprehension, scientific reasoning, and pattern recognition. The primary educational objective is to help students practice and internalize curriculum aligned competencies.

The software will present categorized puzzles with increasing difficulty levels, guide users with clear instructions and optional hints and allow students multiple attempts at completion.

The software will not function as a full learning management system, will not produce grades, and will not require specific knowledge outside of expectations detailed by the Mass. DOE standards.

1.3 Definitions, acronyms, and abbreviations

- **Puzzle:** A unique instance that presents an educational challenge requiring reasoning or subject knowledge.
- **Level:** A segment of the game consisting of a map and its associated puzzles.
- **Map:** The in-game world presented to the player contains game-objects and access to each puzzle.

- **Key:** Rewarded to player after completing a puzzle.
- **Player-character:** The in-game character that the user controls and navigates around the map.
- **UI/UX:** User Interface, the compilation of graphical elements presented to the user within the game.
- **HUD:** [Heads Up Display] Part of the UI, in-game overlay that displays information to the player such as map progress, key progress, prompts, information, hints, and interaction queues.
- **Session:** The period that a user spends playing the game.
- **Hint System:** Game feature that provides partial guidance during puzzles in response to incorrect atte
- mpts at completing the puzzle.
- **Game Object:** A game object is a Unity runtime entity that acts as a container for other functional assets within a scene.

1.4 Organization

Section 2 provides the overall description of the system, including the context in which the game will operate, the characteristics of its users, the target system environment, and the assumptions made during development.

Section 3 describes the specific requirements of the system, covering both the detailed functional requirements and the non-functional requirements needed to meet project goals.

Section 4 outlines the modeling requirements, including use cases, diagrams, class models, sequence diagrams, and relevant interaction scenarios that illustrate the system's behavior.

Section 5 presents information about the prototype, including the current implementation status and examples of user interactions with the system.

Section 6 lists all references and supporting documents used in the creation of the Software Requirements Specification.

Section 7 identifies the point of contact for the project and provides the relevant contact information.

2 Overall Description

This section provides a high-level overview of the educational game, including the context in which it will be used, the primary functions it will perform, and the characteristics of its intended users. It describes how the game fits within the broader learning environment and outlines the technical and academic constraints that influence its design. The section also identifies assumptions made during development, such as the availability of compatible hardware and general user skill levels, and highlights dependencies on core technologies like the Unity game engine. In addition, this section notes requirements that are beyond the current project scope and may be included in future versions of the game. Together, the following subsections establish the foundation for the detailed requirements, models, and prototype descriptions provided later in this document.

2.1 Product Perspective

The proposed system is an educational puzzle game developed in Unity. It is a standalone application designed to run on desktop and laptop platforms. The game integrates multiple subject-based challenges, math, and science, appropriate for middle-school learners. Each puzzle is presented within themed levels or "maps," and players progress linearly or semi-linearly through the content by successfully solving the required challenges. Unity's engine provides the game framework, scene management, UI system, physics, and input handling needed to deliver an interactive and visually engaging learning experience. The game does not rely on external servers for gameplay.

2.2 Product Functions

The system provides several major functions that support the educational goals of the game. First, players interact with subject-based puzzles that are designed around middle-school academic standards, allowing them to reinforce their knowledge in Math, Science, and English through active problem solving. As puzzles are completed, the game unlocks additional levels and content using the GameManager system, enabling linear or semi-linear progression through the themed maps. The system also manages player state by saving and restoring progress, puzzle completion status, earned items, and unlocked content through the SaveSystem. To support learning, the game provides hints, explanations, and corrective feedback when players struggle with specific challenges, helping them understand the concepts rather than simply guessing. In addition, the game includes user interface features that allow players to navigate menus, access settings, and view help screens to support a smooth and intuitive gameplay experience.

2.3 User Characteristics

The primary users of the system are middle-school students, typically between the ages of 11 and 14. These users may have varying levels of reading and comprehension skills, though they are generally familiar with basic computer or mobile device operation. They benefit from clear instructions, visual cues, and intuitive controls that support their learning process. Because students at this age often have different proficiency levels across subjects such as Math, Science, and English, the game is designed to accommodate a range of abilities and provide support where needed.

2.4 Constraints

- The game must run reliably on standard school hardware, which often includes older or low-specification desktop and laptop computers. This imposes limitations on graphic complexity, scene size, memory usage, and background processes, requiring the development team to optimize loading times, reduce asset size, and limit resource-intensive features.
- All puzzles must align with middle-school academic expectations, meaning they must reflect standard learning goals for grades six through eight. These expectations include foundational knowledge in mathematics (e.g., ratios, algebraic expressions), science (e.g., ecosystems, energy transfer), and English (e.g., reading comprehension and vocabulary). Content must be appropriate in difficulty and remain consistent with typical curriculum guidelines.
- Visual and gameplay design must remain accessible and suitable for middle-school students. This includes providing clear instructions, readable text, consistent visual cues, and intuitive controls. The game must avoid themes or visual elements that are distracting, overly complex, or not suitable for learners between the ages of 11 and 14.

2.5 Assumptions and Dependencies

The system assumes that users will have access to a compatible device capable of running Unity-built applications and that they possess basic literacy and digital-navigation skills needed to interact with the game. The project also depends on the continued stability and support of the Unity engine and its associated packages, including the UI toolkit, animation tools, and input system. Additionally, all subject matter content is based on general middle-school curricula, and it is assumed that no region-specific customization will be required unless such changes are added in future releases.

2.6 Apportioning of Requirements

The following requirements are considered out of scope for the current release of Puzzler but may be addressed in future versions:

- Teacher dashboard: A history of the data that each student did in the game, including statistics to show how a class of students may have done on the game. The median scores, highest / lowest scores, and the time each game was completed could be involved here.
- User account system: A system that allows all users to log into their own account, where all accounts are tracked and data is stored. The current version assumes a single local player. This system would allow the developers to make changes to each student's individual account if need be. For example, if a student accidentally trashed an item, the developers would be able to go into that student's account to replace it.
- Online / networking features: Any online leaderboard, cloud save, or teacher-student interaction. The current system operates offline.
- Content expansion and customization: Future maps and puzzles are not included in this version. This prototype only includes 2-3 maps, with 2-3 puzzles per map. Future versions could also allow teachers to customize their own levels for their students, perhaps by inputting their own puzzles. The code would then generate a map that uses data from user input.
- Other platforms: The current version targets Windows desktop via Unity. Future versions of the prototype could include ports to mobile, console, or web.

3 Specific Requirements

1. The system shall provide the user with an interactive educational experience (The game)

A. The game shall provide 2 distinct maps (levels)

i. The game shall include 3 puzzles per map.

a. Each puzzle shall only target one subject

i. Each puzzle shall only target one learning objective in that subject

ii. Puzzle content shall be aligned with 7th grade education standards in these subjects and standards:

i. History and Social Science Framework

a. Pg 100 - https://www.doe.mass.edu/frameworks/hss/2018-12.pdf

ii. Science and Technology/Engineering

a. Pg 60 - https://www.doe.mass.edu/frameworks/math/2017-06.pdf

iii. Mathematics

a. Pg 61 - https://www.doe.mass.edu/frameworks/math/2017-06.pdf

b. Puzzles shall provide the user a text prompt containing information necessary for solving the puzzle

i. This text will be accessible to the user during the entire course of their attempt at the puzzle.

ii. Text will be presented at, or below, an 7th grade reading level

c. Each puzzle shall contain elements the user will interact with to solve

d. Upon completing a puzzle,

i. The user shall be granted an in-game key to indicate puzzle completion

ii. Be presented with a puzzle summary

ii. Subsequent maps shall contain puzzles that are more difficult than previously completed ones. Difficulty can scale according to one or more of the following criteria:

a. Number of steps required to solve

b. Complexity of concepts involved

c. Amount of information provided vs. required

d. Using concepts that require understanding the learning objective from puzzles presented in previous maps

B. The game shall facilitate transition between maps

i. All subsequent maps, will initially be inaccessible to the player

ii. The next map must be ‘unlocked’ by completing all puzzles on current map, thus

earning all puzzle keys

iii. Upon earning all puzzle keys on current map, the game shall allow the user to transition to the next map.

iv. Upon completion of all puzzles on the final map, the game shall allow the user to end the game.

C. The game shall provide a player character with the following features:

i. Capable of movement to traverse the area within the bounds of the map

ii. Capable of interactions with other game objects found on the map

D. The game shall provide interactable game objects that perform the following functions:

i. Act as a barrier that the player character cannot traverse through (i.e. walls)

ii. Act as an initiating event for each puzzle that transitions the game to puzzle Screen.

E. The game shall facilitate saving game progress

i. User will be able to manually create a save file

ii. The system shall automatically create a save file upon completing a puzzle

G. The game save files shall include the following information:

i. Keys unlocked

ii. Maps unlocked

iii. Player location on map

F. The game shall have a Main Menu presented on startup with the following options available:

i. CONTINUE – Allow the user to continue playing the most recent saved game

ii. NEW GAME – Allow the user to start a new play through

iii. LOAD GAME – Allow the user to continue playing from a save file

iv. SETTINGS – Allow the user to adjust certain in-game options

v. QUIT – Allow the user to exit the program

H. The game shall provide a how-to-play/controls summary page upon beginning a new game.

I. The game shall provide a Heads-Up-Display (HUD) overlay that persists during map traversal and puzzle attempts

i. The HUD shall contain areas to present text to the user

a. When a player is within range over interacting with a game object, a prompt will appear informing the player of what keyboard key to use to interact with the game object.

b. During puzzles, hints will be displayed to the user

ii. While attempting puzzles, the HUD shall provide the user access to any materials initially presented (the prompt) or required to complete the puzzle.

iii. The HUD shall contain fields to indicate game progression:

a. Map progression – Display indicators for:

1) Completed maps

2) Current map

3) Remaining maps

b. Key progression – Display indicators for:

1) Placeholders for unearned keys

2) Earned keys

iv. The HUD shall contain interactable icons:

a. To toggle ON/OFF the background music

b. To provide the user a means to request a hint during puzzles

c. To save game progression

d. To return to main menu

e. To view the how-to/controls summary

3. The puzzles shall provide feedback to the user upon submitting their attempted solution

A. The game shall track the number of incorrect attempts for each puzzle.

B. The game shall provide tiered hints after incorrect attempts:

i. After 2 incorrect attempts — give a first-tier hint offering general guidance.

ii. After 3 total incorrect attempts — give a second-tier hint with more specific direction.

iii. After 4 total incorrect attempts — give a third-tier hint giving near-complete guidance.

C. Feedback is provided via the HUD indicating attempt was “Incorrect” or “Correct”.

D. The game shall allow retry attempts without limitation.

E. The game shall prompt the user to “try again” via the HUD after incorrect attempts

4 Modeling Requirements

Use Case Diagram: split into 2 parts for readability

Part 1 shows Game mechanics accessible to the player as part of the main menu, world maps, and HUD buttons. Part 2 shows the internal features and logics related to certain actions the player can perform from part 1.

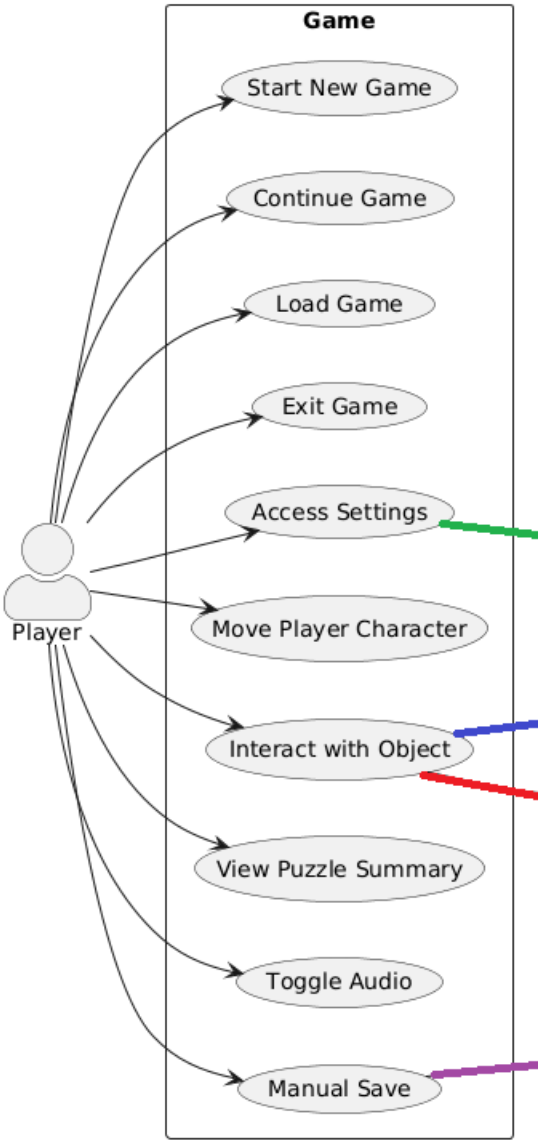


Diagram 1: Use Case part 1

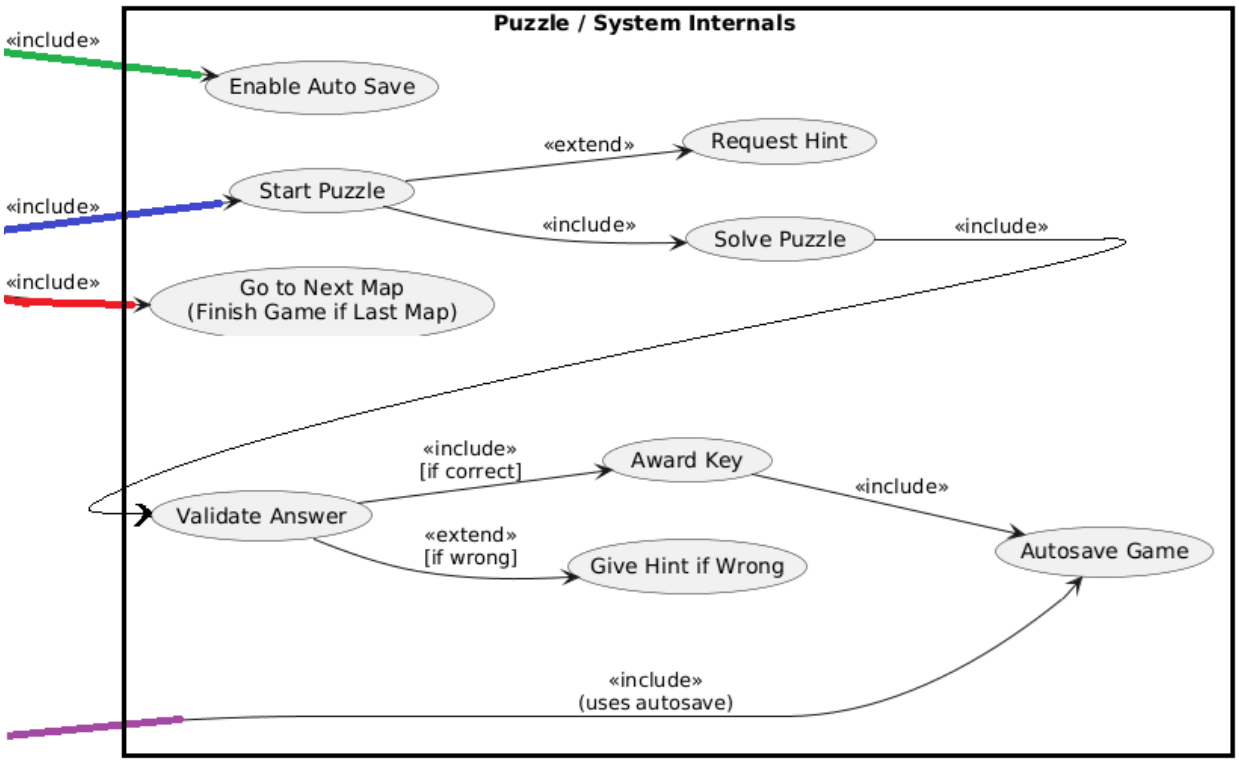


Diagram 2: Use Case part 2

Use Case Name:	Start New Game
Actors:	Player
Description:	The player starts a brand-new playthrough from the main menu, creating an initial game state with only the first map unlocked, and no puzzles completed or keys collected.
Type:	Primary
Includes:	View HUD, Auto-Save Progress
Extends:	None
Cross-refs:	Requirements for game startup, initial state, map unlocking, and saving
Uses cases:	Provides entry into core gameplay use cases such as move player, character, interact with object, solve puzzle, and advance to next map

Use Case Name:	Continue Game
Actors:	Player
Description:	The player resumes an existing playthrough from the main menu when a recent save exists. The system loads the most recent autosave (or last save) so gameplay can continue from where it left off.
Type:	Primary

Includes:	None
Extends:	None
Cross-refs:	Requirements for loading and resuming progress
Uses cases:	Provides entry back into core gameplay, use cases such as move player character, interact with object, start puzzle, solve puzzle, and go to next map.

Use Case Name:	Load Game
Actors:	Player
Description:	The player selects a specific saved playthrough from the main menu. The system loads the chosen save slot, restoring maps, puzzles, keys, position, and settings.
Type:	Primary
Includes:	none
Extends:	None
Cross-refs:	Requirements for save slot management and resuming progress
Uses cases:	Provides entry into core gameplay use cases.

Use Case Name:	Exit Game
Actors:	player
Description:	The player exits the application from the main menu.
Type:	Primary
Includes:	None
Extends:	None
Cross-refs:	Requirements for application exit and resource cleanup
Uses cases:	May follow gameplay or navigation

Use Case Name:	Access settings
Actors:	player
Description:	The player opens the settings menu to adjust gameplay options.
Type:	Primary
Includes:	Enable auto save
Extends:	None

Cross-refs:	Settings and configuration requirements
Uses cases:	Includes enable auto save; may be accessed from main menu

Use Case Name:	Solve Puzzle
Actors:	Player
Description:	The player interacts with a puzzle object on the map, opens the puzzle screen, submits one or more answers, optionally requests hints, and ultimately either solves the puzzle or exits. On the correct solution, the player is awarded a key.
Type:	Primary
Includes:	Validate answer
Extends:	None
Cross-refs:	Puzzle and feedback requirements
Uses cases:	Precedes Validate Answer, leads to Auto Hint (after incorrect), View Explanation, Award Key

Use Case Name:	Enable auto save
Actors:	Player
Description:	The player turns the auto save feature on or off from the settings. When enabled the system will us autosave game at defined trigger points
Type:	Supporting
Includes:	None
Extends:	None
Cross-refs:	Saving
Uses cases:	Included by access settings, controls whether triggers such as award key and manual save use autosave game

Use Case Name:	Move Player Character
Actors:	Player
Description:	The player uses input controls to move the character around the map, navigate to different rooms, and reach interactive objects or exits.
Type:	Primary
Includes:	None
Extends:	None

Cross-refs:	Movement and control requirements
Uses cases:	None

Use Case Name:	Interact with Object
Actors:	Player
Description:	The player interacts with an object in the environment (such as a puzzle portal, chest, sign, or door) to trigger associated behavior like opening a puzzle screen or showing information.
Type:	Primary
Includes:	Start puzzle, go to next map
Extends:	None
Cross-refs:	Object interaction and environment requirements
Uses cases:	Typically follows move character, includes start puzzle for puzzle objects and may include go to next map for exits that are unlocked

Use Case Name:	Start puzzle
Actors:	player
Description:	The player transitions from the map view into an active puzzle view after interacting with a puzzle object. The puzzle interface is displayed, and input for solving becomes available.
Type:	supporting
Includes:	Solve puzzle
Extends:	None
Cross-refs:	Puzzle initiation and UI requirements
Uses cases:	Included by interact with object, includes solve puzzle, may be extended by request hint if the player asks for help

Use Case Name:	Solve puzzle
Actors:	player
Description:	The player works on a puzzle by entering one or more candidate answers and submitting them until the puzzle is either solved or exited.
Type:	primary
Includes:	Validate answer
Extends:	None

Cross-refs:	Puzzle logic, validation, and feedback requirements
Uses cases:	Includes validate answer for each submitted answer, through validate answer may lead to give hint if wrong and award key. Successful award key contributes to conditions for go to next map.

Use Case Name:	Award Key
Actors:	Player
Description:	When the player solves a puzzle correctly, the system grants a key (or equivalent progression token) and updates the player's key count and progression status.
Type:	Supporting
Includes:	None
Extends:	Validate Answer
Cross-refs:	Key progression and unlock requirements
Uses cases:	Triggered as an extension from Validate Answer upon a correct solution; contributes to conditions for Finish Game / Advance to Next Map

Use Case Name:	Validate Answer
Actors:	Player
Description:	The system checks the player's submitted puzzle answer(s) against the correct solution, determines correctness, and records the result for feedback and progression.
Type:	Supporting
Includes:	None
Extends:	None
Cross-refs:	Puzzle validation and scoring requirements
Uses cases:	Included by Solve Puzzle; may trigger give hint if wrong when the answer is incorrect and award key when the answer is correct

Use Case Name:	Give hint if wrong
Actors:	player

Description:	When an answer is incorrect, the system may provide a contextual hint that guides the player toward the correct solution without fully revealing it.
Type:	supporting
Includes:	None
Extends:	Validate answer
Cross-refs:	Hint system
Uses cases:	Triggered as an extension from validate answer when an answer is wrong

Use Case Name:	Award key
Actors:	player
Description:	When the player solves a puzzle correctly, the system grants a key
Type:	supporting
Includes:	Autosave game
Extends:	Validate answer
Cross-refs:	Key progression and unlock requirements
Uses cases:	Extends validate answer upon a correct solution, includes autosave game to persist the updated progression. Contributes to satisfying conditions for go to next map

Use Case Name:	Autosave game
Actors:	player
Description:	The system saves the players current state in response to specific triggers, such as obtaining a key or performing a manual save action
Type:	supporting
Includes:	None
Extends:	None
Cross-refs:	Saving
Uses cases:	Included by award key and manual save

Use Case Name:	Manual save
Actors:	player
Description:	The player explicitly triggers a save from the HUD, the system performs the same saving behavior as autosave game

Type:	supporting
Includes:	Autosave game
Extends:	None
Cross-refs:	Manual saving
Uses cases:	Accessible from he HUD while in game, include autosave game so that the same save pipeline is used for both

Use Case Name:	Go to next map
Actors:	player
Description:	When the player has collected the required keys and reaches an exit game object the system unlocks and loads the next map, if there are no remaining maps the system instead shows a game completion screen
Type:	primary
Includes:	None
Extends:	None
Cross-refs:	Map progression and win condition requirements
Uses cases:	May be initiated via interact with object at an exit, uses player key count to determine whether progression is allowed and whether to move on to another map or finish game

Use Case Name:	Request hint
Actors:	player
Description:	The player may explicitly request a hint from the puzzle HUD interface. The system provides a clue that narrows the search space without revealing the correct answer
Type:	supporting
Includes:	None
Extends:	Start puzzle
Cross-refs:	Hint system
Uses cases:	Extends start puzzle as an optional behavior while working on a puzzle

Use Case Name:	Toggle audio
Actors:	player
Description:	The player turns the game audio on/off

Type:	supporting
Includes:	None
Extends:	None
Cross-refs:	Audio requirements
Uses cases:	Accessible via the HUD during gameplay

Use Case Name:	View puzzle summary
Actors:	player
Description:	The player opens a summary panel that shows information about the current puzzle or map progress
Type:	supporting
Includes:	None
Extends:	None
Cross-refs:	UI
Uses cases:	Accessible via the HUD during gameplay, supports solve puzzle and general progression

Class Diagram:

Here is the layout of classes utilized by the Unity environment to perform game related logics and tasks. Does not include Unity specific development objects such as ‘Scenes’ which are used to create graphical components such as HUD, Maps, and screens displayed in the game but are not composed of code.

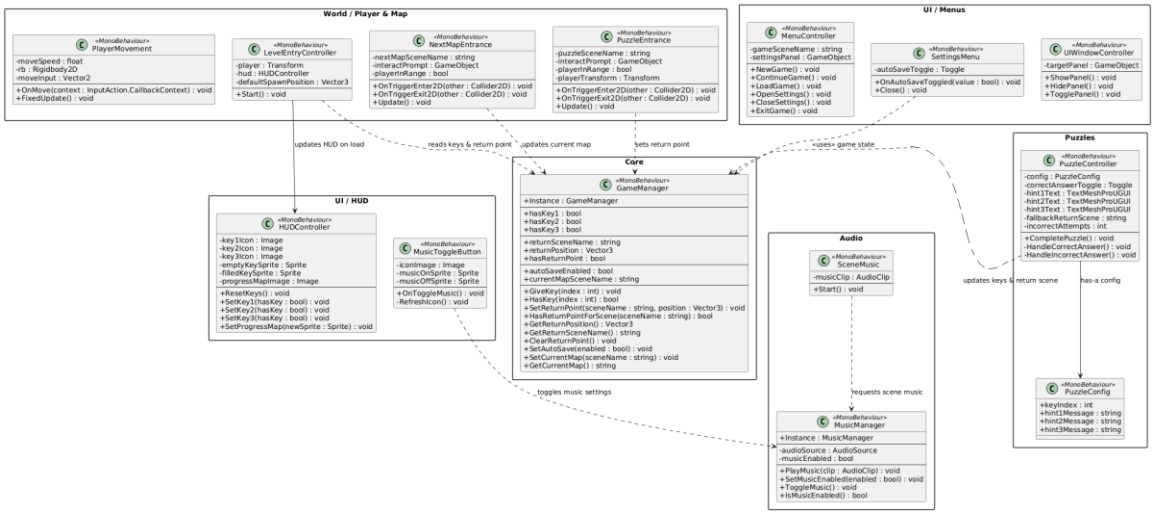


Diagram 3: Class diagram

UI / Menus

Class Name	MenuController
Attributes	GameSceneName: What map scene to load when user presses ‘New Game’ or ‘Continue’ SettingsPanel: panel who’s visibility is toggled to visible upon pressed the ‘Settings’ button. Toggle to invisible when user pressed the ‘X’ close button.
Operations	Handle functionality related to the Main Menu to enable transitioning between panels and scenes
Relationships	None

Class Name	UIWindowController
Attributes	TargetPanel: panel to have it’s visibility toggled.
Operations	Used in the map scenes to toggle the ‘Controls’ panel
Relationships	None

Class Name	SettingsMenu
Attributes	AutoSaveToggle: user may select to enable/disable auto saves
Operations	Control center for what customizable settings the user has selected.
Relationships	None

World / Player & Map

Class Name	PlayerMovement
Attributes	MoveSpeed: player speed Rb: reference object for unity to control placement and movement of player character
Operations	Used by unity to move player character
Relationships	None

Class Name	NextMapEntrance
Attributes	NextMapSceneName: the name of the map scene to load

	InteractPrompt: The text field on the HUD used to communicate interaction possibilities to the user PlayerInRange: a true/false test if the player is close enough to interact with the game object (box collision test)
Operations	Acts a transition between current map and next map
Relationships	GameManager: manages map attributes such as ‘current map’

Class Name	PuzzleEntrance
Attributes	PuzzleSceneName: the name of the puzzle scene to load Interactprompt: the text field on the HUD used to communicate interaction possibilities to the user PlayerInRange: a true/false test if the player is close enough to interact with the game object (box collision test) PlayerTransform: x,y,z location data for where the player character was on the map prior to transitioning to puzzle scene
Operations	Transitions the game from the map scene to the puzzle scene
Relationships	GameManager: manages map attributes such as ‘ReturnSceneName’, ‘ReturnPosition’, and ‘HasReturnPoint’

Class Name	LevelEntryController
Attributes	Player: x,y,z transform, location data for where to spawn a player into a new map, used for returning from puzzle entrance and saving. Hud: communicates with hud DefaultSpawnPosition: if no other player transform provided, a default position is set
Operations	On first spawning in new map, used to setup player positioning
Relationships	GameManager: communicates state variables such as keys earned on this map HUDController: updates HUD with key data

Core

Class Name	GameManager

Attributes	Instance: The gamemanager object being used to control the game (self reference) HasKey1,2,3: Has the player earned the respective key ReturnSceneName: used when transitioning between a map and it's puzzles. ReturnPosition: where to draw the player character upon transition HasReturnPoint: checks for returnPosition otherwise uses default location. AutoSaveEnabled: setting used to control saving after puzzle completion (controlled in settings menu from main menu) CurrentMapSceneName: the name of the current map scene
Operations	Acts as a holder of information that must survive through multiple scene transitions. This data will be used to create save files and to load save files.
Relationships	UI elements to update player accomplishments and game objects that require persistent data through scene transitions

UI / HUD

Class Name	HUDController
Attributes	Key1,2,3Icon: ui element to display empty key placeholder sprite and earned key sprite EmptyKeySprite: used to update key1,2,3Icon, displays that player has not earned that key FilledKeySprite: player has earned this key indicator ProgressMapImage: after completing a map this will be updated to indicate which maps have been completed
Operations	Display relevant progress data to user
Relationships	LevelEntryController: updates scene with user progress data

Class Name	MusicToggleButton
Attributes	IconImage: currently displayed sprite MusicOnSprite: displayed when user enables music MusicOffSprite: displayed when user disables music
Operations	Manages transitions for the music icon on the HUD

Relationships	MusicManager: tells manager if music should be played

Audio

Class Name	SceneMusic
Attributes	MusicClip: the selected audio clip to play on this map
Operations	Used to interface a scene with the music manager
Relationships	MusicManager: provides what music to play to the manager

Class Name	MusicManager
Attributes	Instance: self-reference AudioSource: Unity object that plays the music
Operations	Manages what music is playing and if it should be playing
Relationships	MusictTggleButton: should music be playing SceneMusic: what music is to be played

Puzzles

Class Name	PuzzleController
Attributes	Config: script that houses data for the puzzle such as what key will be awarded by this puzzle and what the text for the hints will be. CorrectAnswerToggle: temporary feature Hint1,2,3Text: The text to give the user when requesting hints/being given hints FallbackReturnScene: the scene to return to after leaving puzzle scene IncorrectAttempts: tracking variable
Operations	Pass the unique puzzle elements to the puzzle scenes through this generalized interface
Relationships	PuzzleConfig: where the data for the puzzle comes from GameManager: communicates return scene name

Class Name	PuzzleConfig
Attributes	KeyIndex: the key to award for completing the puzzle Hint1,2,3Message: text of the hints to provide the user
Operations	Data structure
Relationships	PuzzleController: passes data to this interface

Sequence 1:

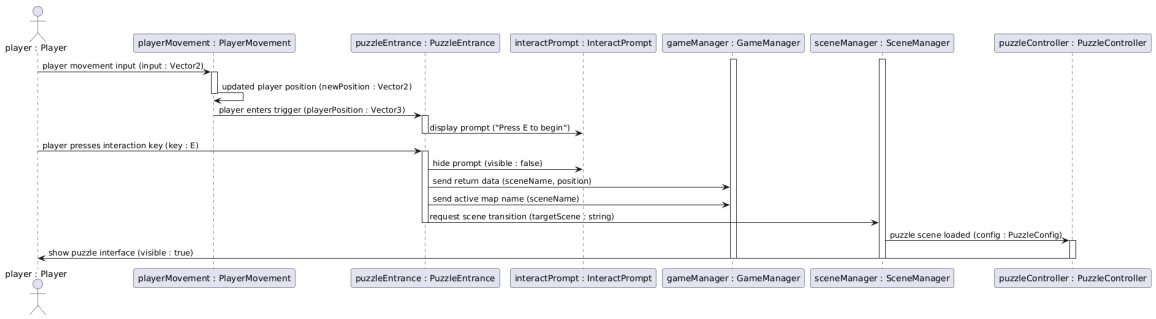


Diagram 5: Enter puzzle sequence diagram

Description: Player enters a puzzle.

The user moves the player character onto the puzzle entrance icon. A prompt is then displayed on HUD “Press E to begin puzzle...” to inform the user of how to proceed. The user presses the E key. The prompt is removed from the HUD and the GameManager saves information about the current map and where the player character is located on it.

Puzzle Entrance contains what scene to load to launch the puzzle and this operation is executed via Unity's scene manager.

Sequence 2:

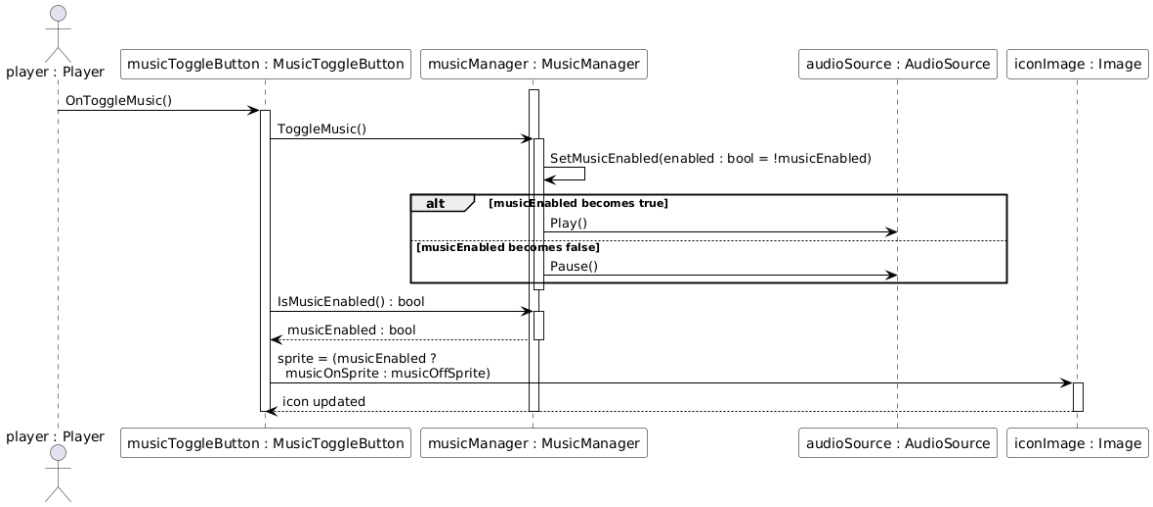


Diagram 6: Music play/pause toggle button sequence diagram

Description: Here the player presses the music control button located on the HUD to either play or pause the background game music. This action is handled by the MusicManager script which toggles the boolean control variable 'musicEnabled'. This then updates the audioSource script, which is a unity specific control module for controlling music and audio sources and their states. Based on the resultant state of this boolean the icon the user pressed is updated to reflect the state of the music as enabled or disabled.

State Diagram:

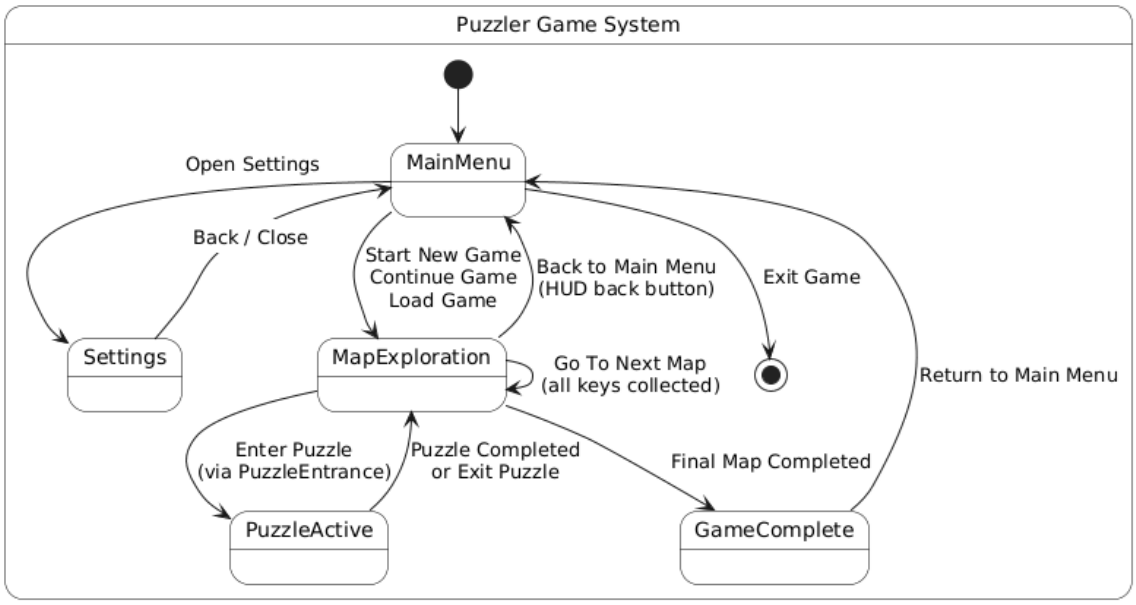


Diagram 7: State diagram

This state diagram represents the flow of an educational game prototype built in Unity. The game starts at the Main Menu, which contains the main screen with game options and a settings panel. From the Main Menu, players enter the first map when starting a new game or return to whichever map they left off at when continuing/loading game.

Each map contains a starting area and three puzzle scenes. Players can navigate from the starting area to any puzzle and return to the starting area after completing it. When ready, players walk to a "next map icon" to progress to the next map, which follows the same structure. On the last map the next map icon ends the game. Players can return to the Main Menu from any map at any time or quit the game from the Main Menu.

5 Prototype

The prototype consists of a basic implementation of the major components and functionality of the intended end product. It begins with the main menu which displays the options available to the player related to how to play the game: New Game, Load Game, or Continue (the last saved game). There is also the ability to Exit the application as well as adjust settings in the Settings menu. Currently only ‘Enable auto save’ is available in the settings menu.

Once the player is in the game there is a simplified map containing icons on the ground that the player can interact with once they have navigated the player character to stand on top of the icons. These icons are a purple question mark to indicate a puzzle which prompts the user with text on the HUD and a green arrow to proceed to the next map once the player has collected the 3 keys for completing each puzzle.

Some puzzles are included for Math and science to indicate the intent of the puzzle feature to be an interactive space that provides feedback / hints and gives the user the opportunity to experience the functionality related to puzzle interactions.

The HUD contains some functional components including: Back, Save, view puzzle, request hint (while in a puzzle), and toggle music. The back button allows the user to return to the Main menu and save allows for a manual save. toggle music disables/enables the scene background music. Finally, the Hint button is a manual request for feedback related to the puzzle that the user would receive had they incorrectly answered the puzzle.

5.1 How to Run Prototype

Prerequisites

- Unity Hub installed on your system (download from unity.com if needed)
- Unity Version 6000.2.10f1 installed through Unity Hub

Installation Steps

- Navigate to the project's GitHub repository at Project Repo.
- Clone the repository using command: `git clone https://github.com/DeepOceanCobaltBlue/SWE_final.git` or alternatively download the project as a ZIP file using the "Code" button on GitHub and extract it to a local directory.
- The project folder structure should contain an Assets folder and Packages folder at the root level

Opening in Unity

- Launch Unity Hub.
- Open Unity Hub and click "Add" or "Open" to select a project from disk
- Navigate to and select the root project folder (SWE_final) that contains the Assets folder.
- Unity Hub will display the project with Unity version 6000.2.10f1
- Click on the project to open it in Unity Editor
- Initial project import may take 5-10 minutes as Unity compiles and imports assets.

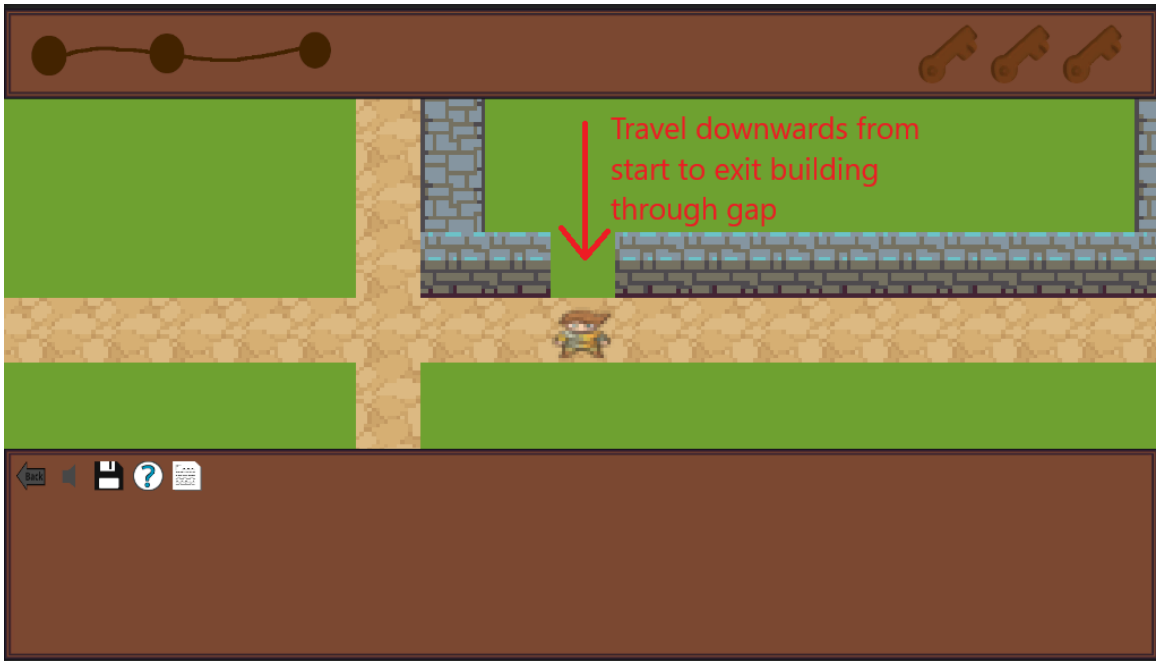
Running the prototype

- Once the project loads in Unity Editor, navigate to the Project window at the bottom
- Locate the Scenes folder within the Assets directory
- Double-click the Main_Menu scene
- Click the Play button at the top center of the Unity Editor to run the game
- Use WASD for movement through the scene and E to interact
- Click the Stop button to stop the scene and return to edit mode

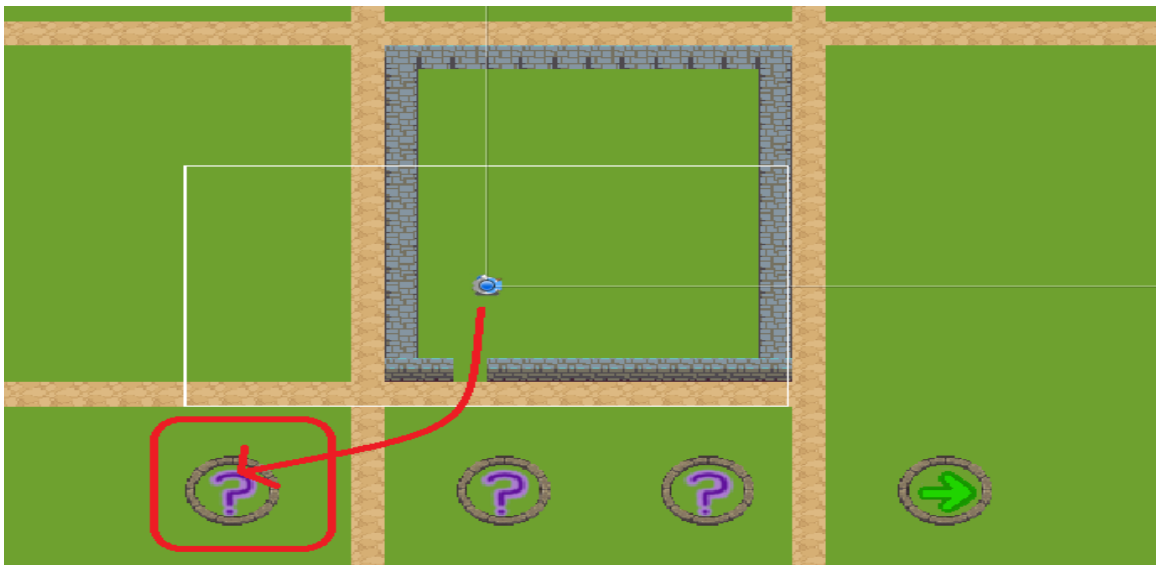
5.2 Sample Scenarios

[1] The main menu is displayed, press ‘New Game’ button to begin

[2] The starting map has been loaded with the player character at the center within a destroyed building. Use WASD to navigate downwards to leave through the opening in the walls.

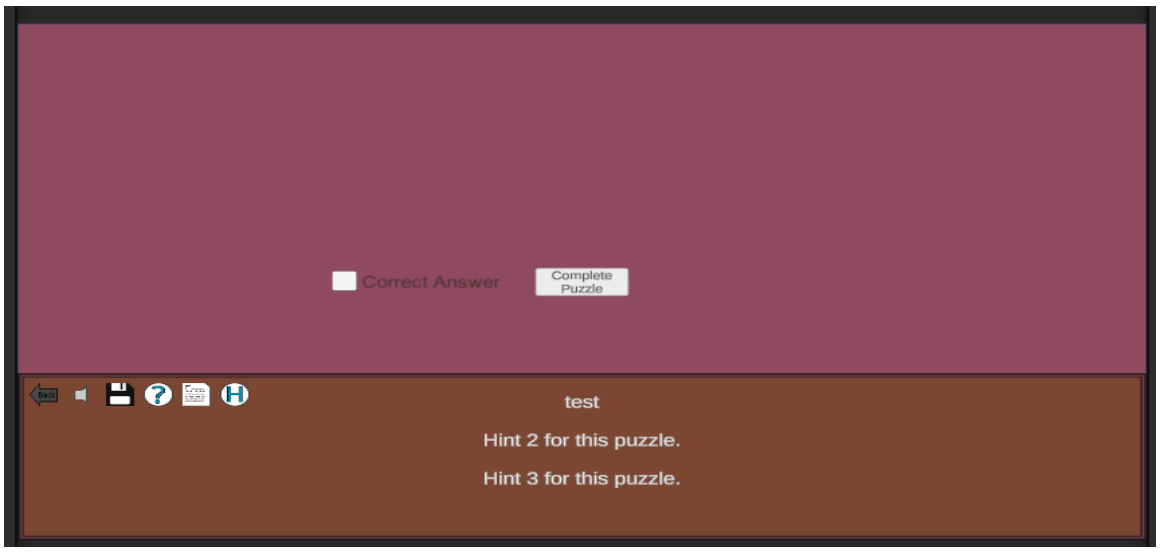


[3] Here is a birds-eye view of where the player begins and where the first puzzle is located. Approaching any puzzle is possible as they are not ordered, this one was chosen for the scenario.



[4] Here the player character is standing at the entrance to the first puzzle. The HUD contains the prompt ‘Press E to begin puzzle...” and queues the user to interact via the ‘E’ key on the keyboard to start the puzzle.

[5] Here the puzzle screen is displayed. This is a placeholder for puzzles but displays all the mechanics. On the HUD all three hints are displayed as well as the options icons. Here the icon of a blue H in a white circle is how the user can manually request a hint. Otherwise they are provided after an incorrect answer. Hitting submit without checking ‘correct answer’ will simulate incorrect answers and grant hints on attempt 2,3, and 4.



[6] Here the user has completed the puzzle and been returned to the starting map. The key granted for completing a puzzle is highlighted in red at the top right of the HUD.

6 References

D. Thakore and S. Biswas, “Routing with Persistent Link Modeling in Intermittently Connected Wireless Networks,” Proceedings of IEEE Military Communication, Atlantic City, October 2005.

5 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.