

Contents

[1 Introduction 3](#_gjdgxs)

[1.1 Purpose of this document 3](#_30j0zll)

[1.2 Scope of the development project 3](#_1fob9te)

[1.3 Definitions, acronyms, and abbreviations 3](#_3znysh7)

[1.4 References 3](#_2et92p0)

[1.5 Overview of document 3](#_tyjcwt)

[2 System architecture description 3](#_3dy6vkm)

[2.1 Overview of](#_1t3h5sf) modules/components 3

[2.2 Structure and relationships 3](#_4d34og8)

[2.3 User interface 3](#_17dp8vu)

[2.4 User interface issues 3](#_3rdcrjn)

[3 Detailed](#_26in1rg) descriptions of components (ONLY 2 ARE REQUIRED) 3

[3.1 X Component (or Class or Function ...) 3](#_lnxbz9)

[3.2 Y Component (or Class or Function ...) 4](#_35nkun2)

[4 Reuse and relationships to other products 4](#_1ksv4uv)

[5 Design decisions and tradeoffs 4](#_44sinio)

[6 Pseudocode for components 4](#_2jxsxqh)

[7 Appendices (if any) 4](#_z337ya)

[Software component template for section 3 4](#_3j2qqm3)

# Introduction

## Purpose of this document

Full description of the main objectives of the SDS document.

## Scope of the development project

This will be similar to what was written in the SRS.

## Definitions, acronyms, and abbreviations

Be sure to alphabetize!

## References

This section will include technical books and documents related to design issues. Be certain that the references you give are complete and in the appropriate format.

## Overview of document

A short description of how the rest of the SDS is organized and what can be found in the rest of the document. This is not simply a table of contents. Motivate and briefly describe the various parts!

# System architecture description

## Overview of modules / components

This subsection will introduce the various components and subsystems.

## Structure and relationships

Make clear the interrelationships and dependencies among the various components. Structure charts can be useful here. A simple finite state machine can be useful in demonstrating the operation of the product. Include explanatory text to help the reader understand any charts.

## User interface







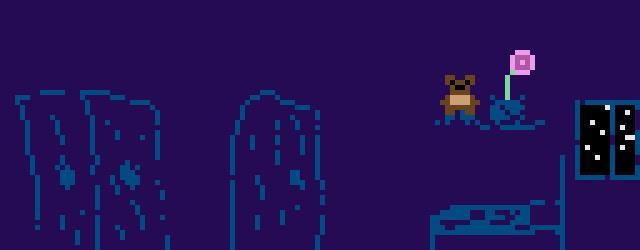








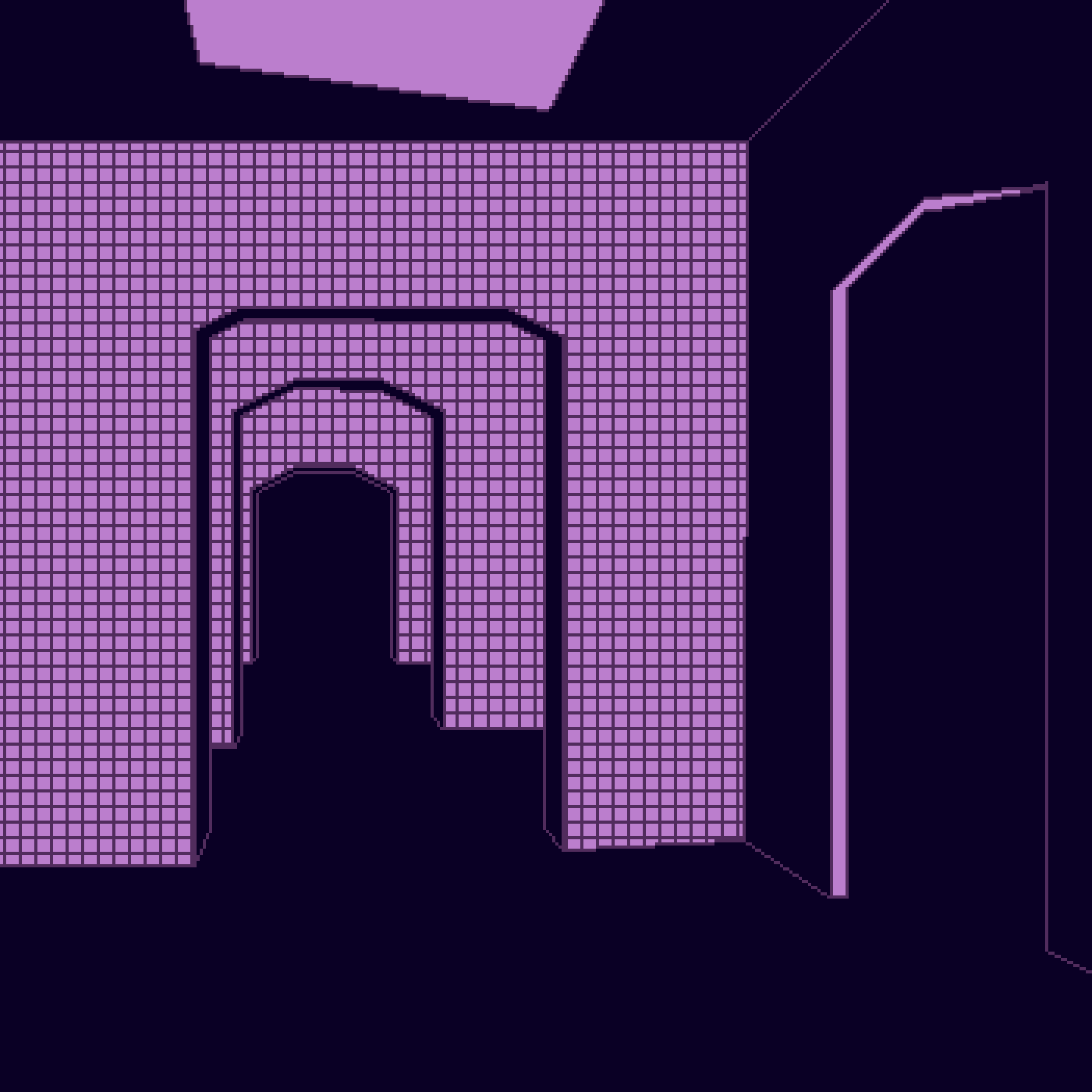














## User interface issues

A graphical user interface issue may occur anytime a user performs an action within the game that the project team did not implement or did not provide functionality for. An example of a type of message that would appear to the end-user in this scenario could be seen in the graphic “Figure 6: Error Message Prototype.”

# Detailed description of components (ONLY 2 ARE REQUIRED)

## MenuSystem Function

| Identification | MenuSystem. Located in the system’s file directory. |
| --- | --- |
| Type | Class |
| Purpose | The MenuSystem will need to successfully implement three options that the end-user can choose from. These options will be play, settings, and quit. The MenuSystem will also need to possess mouse and keyboard functionality, which will allow the user to be able to select the option they desire. In terms of the options, the “Resume” button will need to possess functionality that forces the game to start when selected by the end-user. The “Settings” button will need to possess functionality that displays a secondary menu once selected by the user. The secondary menu will have options to adjust the sound output of the game, these options will be “music” and “sound effects.” Once the user makes changes, to their sound output, these changes should be reflected once the game is resumed. Finally, the “Quit” option will return the end-user to the title screen. |
| Function | This component will display a menu to the user and allow them to choose one of the previously discussed options. This component will allow the user to navigate within the game after pausing. The transformation process will take place once the user has started the game and pauses it. Once the end-user pauses the game, the system will display a menu to them, which will allow them to continue navigating within the game. The specific inputs that will be processed with this component are keyboard inputs. If the end-user presses the “ESC” key on their keyboard while playing the game, the pause menu will be displayed on their screen. The algorithms that will be utilized for this component will display the pause menu once the system has detected an “ESC” key input from the keyboard of the end-user. The outputs that will be produced from this component are a pause menu that will allow the end-user to navigate and change settings if they wish to do so, as well as “Resume” and “Quit” options. The settings that will possess the ability to be adjusted are the “music” and “sound effects” settings. These data items will be stored on the production teams’ computer systems, or more specifically, in their file directories under the “Eclipse Workspace” folder. Eclipse is the IDE the development team is utilizing for this project. The data items that are modified in this component are the MenuSystem function, which will be modified to reflect the design implementations previously discussed within this component. |
| Subordinates | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| Dependencies | This component’s function and performance interacts with other components within the game, such as Use Case #3, which is titled “Recognition of Keyboard and Mouse.” This component will interact with Use Case #3 because it is required that the end-user presses the “ESC” key on their keyboard in order to pause the game, which will call the MenuSystem function and display the pause menu. If Use Case #3 fails and no keyboard and mouse is recognized by the end user's computer system, then this component will be rendered useless. The MenuSystem component is used for the detection of keyboard and mouse components because the MenuSystem function cannot execute properly if the detection of keyboard and mouse components is not functioning properly. Other components utilize this component because, in order for the end-user to adjust the volume in the settings of the MenuSystem, the adjusting volume component relies on a functioning MenuSystem component. The MenuSystem component is called when the end-user presses “ESC” on their keyboard, first the user presses the “ESC” key on their keyboard, then the MenuSystem causes a pause menu to be displayed to the end-user. The programmers of the development team are responsible for the creation of this function. |
| Interfaces | The external graphical user interface associated with this component will display a pause menu to the user once the “ESC” key is pressed on the end user's keyboard while playing the game. This external interface will allow the end-user to navigate through the pause menu and adjust their sound settings if needed. The internal interface associated with this component will include code within the Eclipse IDE that will prompt a graphical user interface to be displayed to the user that includes three options: “Resume,” “Settings,” and “Quit.” The error messages associated with this component are none. The only way the end-user can encounter an error on the pause menu is if they press any keys on the keyboard other than the “Enter” key and the directional keys. If other keys are pressed, an error message will not be displayed, however, the pause menu will not be able to be utilized. The screen format of the pause menu will possess three options. At the top of the pause menu, the “Resume” option will be displayed. This option will allow the end-user to resume the game once they are finished navigating the pause menu. The second option, which is located below the “Resume” option, will be the “Settings” option. This option will include two additional options titled “Sound Effects” and “Music.” The end-user will be able to adjust the sound output that is associated with these two options. Finally, the “Quit” option will be displayed at the bottom of the pause menu. This option will allow the end-user to exit the current game they are playing and return to the title screen. |
| Resources | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services. |
| Processing | The full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| Data | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary. |

*Table 1: MenuSystem Component*

Examples of the MenuSystem component are:

1. Title Screen
2. Pause Screen
3. Settings Screen
4. Error Screen

## ControllerMovement Function

| Identification | ControllerMovement. Located in the system’s file directory. |
| --- | --- |
| Type | Class |
| Purpose | The ControllerMovement component will need to successfully implement a multitude of functional requirements. The functions that this component will need to implement are forward movement, backward movement, jumping, and implementation that will allow the end-user to perform these movements using a keyboard. These functions will be mandatory in order for the end-user to move and progress through the level of the game. Each of the movement features will possess its own functionality and will be implemented separately into the game. For example, the forward movement feature will be implemented independently of the other movement functions since forward movement will be associated with the right arrow key (->) as well as the “D” key. This process will be duplicated for the implementation of the remaining movement features, which are an essential component in order for the game to function as intended. |
| Function | The ControllerMovement component will control the end user’s ability to perform different movements throughout the level. The three movements that the end-user will be able to perform are forward movement, backward movement, and jumping. The system will take in these inputs from the user via the end user’s keyboard, which should be connected to their computer. The forward movement function will be implemented by allowing the user to press the right arrow (->) key or the “D” key, which will allow the in-game character to move forward throughout the level. This function is necessary because if the character cannot move forward through the level, they will not be able to complete the level. The next movement component to be implemented is the backward movement function. The end-user will be able to move their character backward by pressing the left arrow (<-) key or the “A” key on their keyboard. This maneuver will allow the end-user to move their character backward in case they missed a checkpoint or they are attempting to avoid an incoming threat. The last movement component that the development team will need to implement is the jumping function. This movement function can be performed by the end-user by utilizing either the up arrow (^) key or the “W” key on the keyboard. This maneuver will allow the end-user to hurdle over oncoming threats or jump onto different platforms established throughout the level. The algorithms that are used in the ControllerMovement component will be established by creating the movement functions, then implementing them separately based on the movement type. The outputs that are produced from this component are an in-game character being able to move forward, backward, and laterally within the game’s level. These data items will be stored in the project team’s file directory under the “Eclipse Workspace” folder. These data items will be modified within the Eclipse IDE, which is the software the development team is using for this project. |
| Subordinates | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| Dependencies | Implementing the ControllerMovement component will be vital for the project team in order to obtain a successful end product. This component is essential because many of the games’ other features will rely upon the end-user being able to navigate through the level at their leisure. The ControllerMovement component only relies upon an implemented character design, the implemented level, and the end-user possessing a functional keyboard with the necessary keys. Meanwhile, many other of the games' features and functions will rely on the ControllerMovement component. For example, in order for a checkpoint to be reached and a save to be initiated, the end-user must reach a certain point in the level where the system will recognize the user’s progress as sufficient enough for a save. This will allow the user to not have to start from the beginning of the level upon reset of the game. However, this feature is only utilized if the user is able to reach that point in the level, which requires correct implementation of the movement mechanics. Another feature that relies upon the ControllerMovement component is the game’s camera and parallax scrolling. These features rely upon the ControllerMovement component because if the in-game character is unable to move, then the camera and scrolling features will not function properly or be unutilized. As long as the character is progressing through the level, the camera and scrolling features will follow the character and function as intended. This also means that the timing of the camera and scrolling features must align with the ControllerMovement component, which will allow the end-user to have an interactive and enjoyable experience. The programmers of the development team are responsible for the creation of this function. |
| Interfaces | The external graphical user interface will present the end-user with a character that they are able to control with their keyboard. Once the end-user has loaded into the level, they will be able to control the in-game character by being able to move forward, backward, and laterally. This external interface will combine the ControllerMovement component with the games’ other functions to provide an enjoyable experience for the user. The internal interface associated with this component will include code within the Eclipse IDE that will allow the user to move forward, backward, and laterally throughout the level. No error messages will be associated with this component. If the end-user incorrectly utilizes this component by pressing the wrong keys on the keyboard, then the game will not be able to progress. As a result of incorrectly using this component, the end-user will be unable to experience the game in its entirety or the games’ other features. There is no screen format to this component, instead, this component is controlled entirely by the end-user and their keyboard. Based on the keys pressed by the end-user, the on-screen character will act accordingly by constantly waiting for inputs from the user. If the end-user is able to correctly perform the functions associated with this component, they will be able to play the game as intended by the development team. |
| Resources | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services. |
| Processing | The full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| Data | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary. |

*Table 2: ControllerMovement Component*

Examples of the ControllerMovement component are:

1. In-Game Controller Movement
2. Title Screen Controller Movement
3. Pause Menu Controller Movement

# Reuse and relationships to other products

For teams doing enhancement work, reuse is an important issue. Most enhancement work should focus on extending, rather than replacing, the design and product development from earlier semesters. For teams doing new development, reuse can also be an important strategy. In some cases, there is a freeware that could be incorporated. In other cases, some existing modules or classes could be adapted. Another possibility is the use of special tools that produce open-source results and are thus permissible under the terms of this course.

This section should include the following subsections as appropriate:

* How reuse is playing a role in your product design
* How reuse is playing a role in your product implementation (and the motivation for changes)
* If you are not reusing available material, then give motivation for why it is being thrown out.

# Design decisions and tradeoffs

The menu will be crafted in a simplistic form as the development team only has a few months to develop any features that we deem are necessary for the game to run properly. In later updates, other settings may be added to improve gameplay and the user interface. Gameplay will be based on two-dimensional games as it complements the style and aesthetic of the type of game the development team wants to create. Using original sprite assets and simple gameplay mechanics such as jumping and moving.

# Pseudocode for components

Utilize the use cases to create pseudocode for components.

# Appendices (if any)