Review of Sprint 2 and Proposal for Sprint 3 Backlog

ACS 2025/26 Application Development Project

Group 5

Moritz Lackner

Oskar Lukáč

Dominika Nowakiewicz

Mihail Petrov

Rodrigo Polo Lopez

Tieme van Rees

Product Owner: Vineet Gokhale

Supervisor: Beata Skuczynska

Professional Skills: Renée Tentori

10.10.2025

I. Table of Contents

II. Introduction

This document presents the review of Sprint 2 and the proposal for Sprint 3 of the ACS Application Development Project. It outlines the tasks completed during the previous sprint, including their outcomes and software locations, as well as the new backlog items planned for the next sprint. The goal is to show the progress made in developing the game *Escape of the Nightmare* and to define the priorities and improvements for the upcoming sprint based on team reflection and feedback.

III. Completed sprint 2 backlog

PB 2-1: Quit command

Student: Dominika Nowakiewicz

Status: Completed

Add a quit command that immediately exits the game without saving.

Acceptance criteria:

- Typing quit from any room terminates the program.

Software location basic.command.py

PB 2-2: Pause (Save) & Resume on Next Start

Student: Mihail Petrov

Status: Partially completed

Short description: Implemented full game state saving in a SQLite3 Database. Successfully restores player state (name, current room, state dictionary, inventory, time).
However, the current implementation stores everything inside of the state dictionary in a single JSON file, which does not comply with the normalised structure of the database recommended by the teachers(separate entities for every state file member).
To be refined in Sprint 3 for proper data structure and separation.

PB 2-3: status command

Student: Oskar Lukáč

Status: Completed

Short description: Show the player's name, elapsed play time, and the percentage of rooms finished.

Software location

basic_commands.py

PB 2-4: Scoreboard

Student: Oskar Lukáč

Status: Completed

Short Description

Implement a persistent leaderboard system to track and display the top player attempts. The leaderboard records both completion percentage and play time, allowing players to compete for the best performance. Load and save scoreboard.

Acceptance Criteria

- The “scoreboard” command prints the top 5 attempts in ranked order.
- Leaderboard entries are sorted by:
  1. Highest completion percentage (descending)
  2. Shortest play time (ascending)
- The leaderboard is updated on pause, quit, scoreboard view, and game finish.
- Data is persistent between sessions using a file-based or database solution.

Design Choices

- File-based storage
- Integrated with existing game lifecycle events.

Software Location

main.py
scoreboard.py (new module)
data/scoreboard.json (persistent storage)

PB 2-5: Timer

Student: Tieme van Rees

Status: Completed

Short description: Pausing is available in every room and the time freezes when the command “pause” is ran. When the game is loaded up from a save state the time starts of where it was.

Software location

Main.py

Basic_commands.py

PB 2-6: Game name, title-screen and end-screen

Student: Moritz Lackner, All developers

Status

Completed. The whole group decided to name the game “Escape of the Nightmare”. A title screen was added with the title in a nice ASCII font and options to start the game. Moreover, there is an option to load a save and display the credits. A end screen was added which is displayed after you finish the game and credits which lists all the developers.

Software location

game/screens.py: whole file

main.py: small changes, added call to title screen and end screen

PB 2-7: Interconnecting rooms

Student: All developers

Status

Partially completed. All rooms use the central inventory to store items and save the state of the room.

Bottlenecks: Because many rooms were still work in progress during the sprint, connections between them was not feasible. This will be addressed in the next sprint.

Software location

All room files respectively

main.py: state dict

PB 2-8: Unifying basic commands

Student: Moritz Lackner

Status

Completed. The general game structure was modified to allow the execution of basic commands being available in each room. To achieve this the functions for the basic commands were collected and moved to a separate file. Also, the game now consists of a single game loop that handles all the user inputs instead of while loops in every room. To adhere to this process all rooms had to be modified. Each room consist now of a

function that is called when the room is entered and a function that checks if a room specific command was entered. These two functions per room are stored in a dictionary in the main file to be called.

Some of the basic commands of the game still need additions in each room. For example, the help screen should show room specific commands. Therefore, specific rooms can add to the basic command by having a separate function for this command. If the command is typed in by the user, both functions will be called.

The room specific command function can also return “go back” to force the player out of the room.

The entry function of each room can also stop the player from entering, if it returns false. Therefore, it is for example, possible to check for specific items, for instance a key to let the player into the room.

To better order the game files a new folder “game” was created to store all files that are relevant to the whole game. This includes the basic functions and the utils file. The “rooms” folder now only lists the python files for the specific rooms.

Challenges: Restructuring the game structure came with the benefit of easier adjusting functions that are used over all games. However, every room file had to be adjusted to make this work. Therefore, a lot of coordination was required to update each individual file of different developers.

Software location

By Moritz Lackner

main.py: room_functions dict, main while loop

game/basic_commands.py: Collected all previous coded basic functions in this file

rooms/classroom_2015.py: added _enter and _commands function

rooms/project_room_3.py: added _enter and _commands function

rooms/study_landscape.py: added _enter and _commands function

All developers:

Changes in each room file respectively: add to functions <room>_enter, <room>_commands

main.py: room_functions dict

PB 2-9: Improving room layout and add map

Student: Moritz Lackner

Status

Completed. An ASCII map of the layout was created to represent each room. When the "map" command is entered the map is shown with a "X" marking the current position of the player. Also, the names of all rooms that can be accessed from the current location are printed in the terminal.

The rooms that can be accessed from each room are stored centrally in the main.py to easily adjust the layout of rooms, add more rooms or add more exits between rooms.

Three basic corridors were added to conform to the map layout.

To better test the game the "admin go" command was added to allow the player to jump to each room. This allows for easy testing and is not intended as a game play feature.

Software location

game/basic_commands.py: show_map, handle_admin_go

main.py: exits in state dict

e_w_corridor.py: whole file

n_s_corridor.py: whole file

lab_corridor.py: whole file

PB 2-10: Polish for dragon room:

Student: Moritz Lackner

Status

Completed. The centralized state variable is used to save all progress in the dragon room and the centralized inventory to store items. Some items are removed if they are used by the player. The dialog is stored in a separate json file and loaded in when entering the dragon room. The room was formatted to conform to the new structure of the main.py file.

Software location

dragon_room.py: small changes

Small changes: main.py, rooms/__init__.py adjusted files to conform with the new internal game structure.

PB 2-11: Improving Riddle room

Student: Oskar Lukac

Status: Completed

Short Description

Enhance the Riddle Room challenge by adding a hint system. This improvement focuses on player guidance after incorrect answers, improving the user experience and reducing frustration.

Acceptance Criteria

- After a failed solve attempt, a hint “Hint: Type solve <answer> to try again.” is displayed.
- The hint shows consistently on repeated failures.
- No existing Riddle Room mechanics are broken.

Design Choices

- Simple and non-intrusive hint to support players while keeping the challenge.
- Integrated into existing solve-check logic.
- Consistent with the minimalist text-based UI style.

Software Location

main.py (hint integration)
rooms/__init__.py (Riddle Room logic updated)

room/riddleroom.py

PB 2-12: Further development Computer lab

Student: Dominika Nowakiewicz

Status: Being worked on

You are now able to interact with the laptop in the room, fixed a bug where the student repeats the same question, you can now interact and finish one puzzle about Intercultural Collaboration, the said questions are randomized, came up with an idea where you have to finish each folder in order to obtain a message to get access to the secret folder

Software location

main.py, rooms/__init__.py, rooms/computerlab.py

PB 2-13: UX polishing in Cloud Room

Student: Mihail Petrov

Status

Completed

Short description: Fix the User Experience by changing the handlers for answers, the question difficulties and the overall view of the room.

Changes in cloudroom.py

PB 2-14: Data-driven questions in Cloud Room

Student: Mihail Petrov

Status

Not completed yet.

Short description: To be split into two separate tasks for Sprint 3 (JSON-based quiz and OpenAI-generated quiz).

PB 2-15: Further development control room

Student: Rodrigo Polo Lopez

Status: Incomplete

Room is enterable.

Software location

Controlroom.py

PB 2-16: Further development Cyberroom

Student: Tieme van Rees

Status:

Partially completed

Short description: The structure of code is all in functions so it's more readable.

Software location

Small changes: main.py, rooms/__init__.py adjusted files to conform with the new internal game structure.

IV. Proposal sprint backlog for Sprint 3

PB 3-1: Achievement System

priority: Medium

Student: Oskar Lukáč

- Add an achievement tracker with badges/titles (e.g., “First Solve”, “Perfect Run”).
- Achievements displayed via a new achievements command.

PB 3-2 : Expanded Scoreboard Features

priority: Medium

- Add player names/initials to leaderboard entries.
- Allow viewing full leaderboard history.

PB 3-3: Interconnecting rooms

Origin: PB 2-7

Student: All developers

Priority: High

The goal of this backlog item is to further interconnect the different rooms. This can come in form of progression priority with locked doors and shortcuts between the rooms. There is also the opportunity to add items that can be used in different rooms.

Acceptance criteria:

- Add items that can be used in different rooms
- Add shortcuts between rooms
- Plan player progress through the game
- Add requirements for different rooms to enter (keys)

PB 3-4: Polish UI

Origin: PB 2-6

Student: Moritz Lackner

Priority: Low

Polish UI throughout the game. Make the options clear to the player and provide a uniform look to the game. This should be true for all screens and menues.

Acceptance criteria:

- Make sure UI is clear throughout the whole game

PB-3-5: Further development Computer lab:

Student: Dominika Nowakiewicz

Short description: Code that you can obtain an item after u finish everything in the room, finish the other three folders representing ACS seminars, write a sequence where after failing to answer the questions correctly 3 times you're blocked from the laptop for max 1 minute

Acceptance criteria:

- Make sure the room shows some improvements

PB-3-6: Further development CyberRoom:

Student: Tieme van Rees

Origin: PB-2-16

Short description: Make the room more complex so it's not completed that easily.

Acceptance criteria:

- NPC
- More math problems

PB 3-7: Normalized Save/Load System

Student: Mihail Petrov

Origin: Continuation of PB 2-2

Priority: High

Short description:
Refactor the current saving system to store game state in a normalised structure (e.g. database tables for different attributes of the state dictionary) to comply with teacher feedback.

Acceptance criteria:

Save data split logically into multiple tables with different keys and attributes.
Loading process correctly restores state consistency.

PB-3-8: JSON-based Cloud Room Quiz

Student: Mihail Petrov

Origin: Split from PB 2-14

Priority: High

Short description:

Move Cloud Room quiz questions to a JSON file that can be easily edited or extended without changing code.

Acceptance criteria:

-On room load, questions are read from cloudroom_questions.json.

-If file is missing/invalid, fallback to the static quiz.

-Structure of JSON supports multiple-choice format (question, options, correct answer).

-Basic validation and error handling included.

PB-3-9: OpenAI-generated Cloud Room Quiz (optional stretch)

Student: Mihail Petrov

Origin: Split from PB 2-14 (stretch goal)

Priority: Medium

Short description:

Integrate OpenAI API to dynamically generate quiz questions for the Cloud Room when an API key is provided.

Acceptance criteria:

-If OPENAI_API_KEY exists → generate quiz questions via OpenAI.

-If not → fallback to the JSON-based quiz.

-Questions follow the same structure as existing ones (3 multiple-choice questions).

-Includes simple error handling for failed API requests.

PB-3-10: Contolroom development

Student: Rodrigo Polo Lopez

Origin: PB-2-15

Priority: Medium

Short description:

I want to add an object as a reward for completing the room that you can keep in a common inventory between all the rewards of every room and add punishments in case you fail the puzzles, for example if you get the answers wrong more than twice you won't get access to the reward or something like that.

Acceptance criteria:

-room completion reward

-fail punishments

V. Retrospective

Sprint 2 marked a major step forward in both the technical development and teamwork of Group 5. Due to many new tasks in addition to leftover tasks from the previous sprint the workload was bigger and more varied than in the previous sprint. Moreover, the tasks were more interconnected between different files and there were dependencies on others group members' work. However, the team handled the challenge and have finished most of the backlog. This led to noticeable improvement in organization, planning, and execution. Most members began working more independently while still maintaining good communication and alignment with the team. The introduction of shared systems such as the unified command structure, the timer, pause and resume functionality, and the scoreboard helped bring the game together into a more cohesive and playable experience which helped the team to better envision a final product. This shared vision helped the team to narrow down new suggestions and ideas to expand the game and polish features in the final sprint.

Throughout the sprint, the team placed greater emphasis on quality. Room files were restructured to fit the new architecture, and several shared features were centralized to make future development easier. This not only improved the technical quality of the codebase but also encouraged collaboration between developers, as everyone had to coordinate changes to maintain compatibility across the game's modules.

While communication improved, some challenges remained. A few tasks took longer to complete due to code dependencies or unclear ownership at certain points. The team also noticed that starting some features later in the sprint reduced the opportunity for testing and feedback. The delay can be tracked back to waiting on other features to be completed, because they were dependent on them. Therefore, the team came to the conclusion to better plan the priority of tasks. Moreover, the group will stick more strictly to the assigned priority to ensure features with dependencies are completed earlier in the sprint. In general, these insights led to a stronger understanding of the importance of early planning, regular progress checks, and quick communication when issues arise.

For Sprint 3, the team agreed on several key action points to strengthen workflow and collaboration:

- Begin project work earlier to allow multiple iterations and feedback rounds.
- All group members should stick more strictly to the assigned priority when choosing a task to work on
- If a team member fails to contribute or collaborate after two stand-up meetings, the group will consult the supervisor for next steps.
- Apply feedback methods learned from intercultural collaboration sessions to improve communication.
- Organize a social activity to build empathy and strengthen team bonds.
- Ensure everyone writes a short reflection at the end of the sprint in Google Docs.

Overall, Sprint 2 was a productive and insightful phase for the project. The team became more coordinated, the codebase more unified, and the game itself began to take shape as a connected and engaging experience. With these improvements and action points in place, Sprint 3 is expected to bring even stronger collaboration and further polish to *Escape of the Nightmare*.