ELECTRICAL AND INFORMATION ENGINEERING
University of the Witwatersrand, Johannesburg
Software Development II

Project 2025 — Dig Dug

1 Introduction

Dig Dug was a popular arcade game created and released by Namco in 1982 [1]. The object of the game is to dig tunnels and destroy underground monsters by inflating them with a harpoon-like weapon or by dropping rocks on them. Bonus points can be earned by collecting fruit and vegetables which appear at the centre of the screen. To understand how the game works in more detail, you should watch it being played.

Your task is to write a "Dig Dug" type game for the PC. This implies that your game is based on the same game mechanics as Dig Dug (see Section 4) but you are free, and encouraged, to choose any game theme and backstory that you like.

2 Project Outcomes

On completion of this project you should be able to:

- Perform an object-oriented decomposition/analysis of a software problem which involves a variety of interacting objects.
- Design and implement an object-oriented solution in C++.
- Understand and use existing software libraries in conjunction with your own code.
- Provide a test suite verifying the correctness of your software.
- Provide the requisite documentation for a software project, including an automatically generated technical reference manual.
- Work successfully in small team to deliver a software product.

The purpose of this project is to learn software design and engineering by applying the principles and practices covered in this course to the development of a software product.

3 Constraints

The game needs to be coded in ANSI/ISO C++ using the `raylib-cpp v5.5.0` library. This library is a C++ wrapper for `raylib`. You *may not use* `raylib` *itself*. All submissions must build and run on Windows even though the libraries themselves are cross-platform.

The emphasis of this project is on *good object-oriented design* and not on fancy graphics. Good graphics are only regarded as a minor feature enhancement. With regard to the graphics, the dimensions of your game window must not exceed 1600×900 pixels.

You may not use any other libraries, or frameworks, that are built on top of `raylib` or `raylib-cpp`.

4 Game Mechanics

A screenshot from Dig Dug is depicted in Figure 1 and described below. In order to see the screenshot in colour you should download the project brief from the course website.

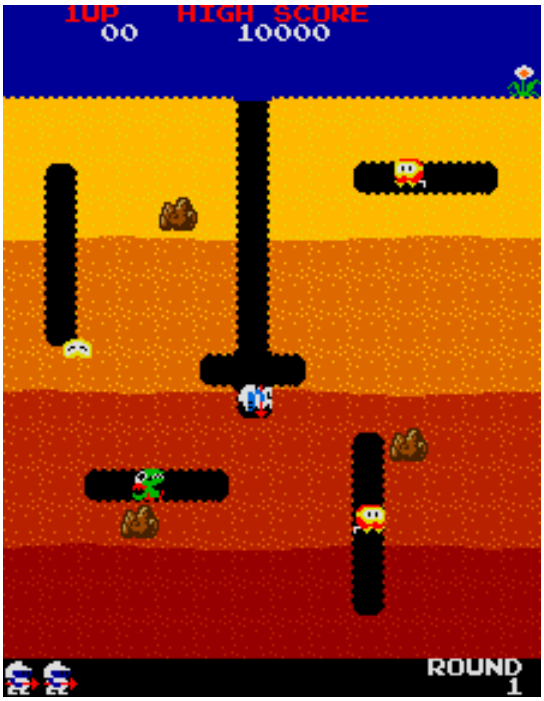

Figure 1: Dig Dug screenshot from [1]

The player, represented by the blue and white Dig Dug character, is near the centre of the screen. Dig Dug can move in four directions (up, down, left and right) and digs a tunnel wherever he moves. He can also fire his harpoon which destroys the underground monsters. His harpoon is fired in the direction that he is facing.

There are two types of underground monsters: red monsters and green dragons. Both of these appear in Figure 1. When these monsters are located within tunnels their movement is restricted by the tunnel; however, they are also capable of floating diagonally through the earth from one tunnel to another. Whenever they do this they turn into a pair of disembodied eyes. A disembodied monster is shown in the left-hand part of Figure 1.

Dig Dug is killed if he collides with either type of monster. He can also be killed by the fire which the dragons occasionally breathe.

Finally, there are a number of rocks scattered about the screen. These rocks will fall if Dug-Dug tunnels underneath them. They can be used to destroy monsters but the player needs to be wary as Dig Dug can also be crushed by a rock if he remains underneath it for too long.

The game level ends when one of the following scenarios occurs:

- Dig Dug wins by destroying every monster on the screen.
- Dig Dug loses by being killed either by a monster or a rock.

5 Categorization of Features

The following section describes the basic functionality that you are required to implement for your game. Additionally, features which are considered minor and major enhancements are listed. If these are implemented well, higher marks will be awarded in the *Functionality* category (see the rubric and Table 1). If you want to implement a feature that is not listed, please contact me first so that I can advise you. You are welcome to include audio in your game; however, this is not regarded as a feature.

5.1 Basic Functionality

The following is considered basic functionality:

- The following game objects exist: Dig Dug, multiple red monsters, and the earth.
- Dig Dug moves correctly through the earth based on player input, and digs tunnels wherever he moves.
- The monsters move autonomously and chase the player. At the start of the game at least one monster is present in the same tunnel as Dig Dug.
- Dig Dug can shoot monsters with his harpoon, and the monsters die instantly.
- The game ends if Dig Dug collides with a monster or Dig Dug shoots all the monsters.

5.2 Minor Feature Enhancements

Minor enhancements include, but are not limited to, the following:

- The red monsters can also change into their disembodied forms and drift from tunnel to tunnel in order to chase Dig Dug.
- Bonus items appear at the centre of the screen and extra points can be earned if Dig Dug collects these.
- Graphics are good (not composed of simple shapes such as rectangles and triangles).
- There is some sort of scoring system and display. High scores are saved to disk from one game to the next, and can be viewed.
- Dig Dug has more than one life and his remaining lives are depicted on the screen.
- The initial locations of the tunnels, rocks, monsters and Dig Dug are read in from a file.

5.3 Major Feature Enhancements

Major enhancements include, but are not limited to, the following:

- Green dragons exist and kill Dig Dug on contact. They breathe fire which is capable of destroying Dig Dug. They can also become disembodied and float from tunnel to tunnel. Note that their disembodied form is different to that of the red monsters.
- Rocks exist and Dig Dug can tunnel underneath these in order to drop them on monsters and crush them. Rocks are also capable of crushing Dig Dug if he stays underneath them for too long.

- The behaviour of Dig Dug's harpoon mirrors the original game more closely. Monsters are not destroyed instantly, instead it takes a few seconds for them to inflate and pop, and Dig Dug is incapable of moving while this is happening (and he is vulnerable to attack by other monsters). Dig Dug may prematurely stop inflating a monster, that is, stop before it pops. In these cases the monster deflates and continues to move around as before.

6 Project Submissions Overview

Agile methodologies are a popular method of developing high quality software. They are both *iterative* and *incremental* in nature. Iterative implies that multiple passes through the phases of the software development lifecycle take place. On completion of each iteration a working build is produced that has a subset of the total functionality that is required. This is usually released to the customer in order to elicit early feedback. Each iteration results in an increment in functionality and iterations continue until the final solution is achieved. Note that your code will probably change significantly between releases. This is to be expected as your design converges on a final solution.

In order to encourage this style of development, there will be two interim submissions and a final submission. The deadlines are available on the course website. The first two submissions are submissions of work-in-progress and a reduced set of project deliverables is required. The final hand-in will require the submission of all the project's deliverables. The deliverables required for each submission are summarized in Table 2 and described in more detail in Sections 6.1 and 6.2.

6.1 First and Second Submission Deliverables

Each submission or release is required to have a splash screen which *correctly* informs the user about the keys used for playing the game.

For the first submission you are expected to show exploratory use of the `raylib-cpp` and `doctest` libraries. Very simple game and test functionality is expected: you need to have implemented Dig-Dug who can be moved around underground, digging a tunnel as he moves. There must be some associated tests verifying that Dig-Dug moves correctly and that his movement is restricted by the screen boundaries.

For the second submission you are required to build on the functionality of the first submission. To ensure that you leave yourself sufficient time towards the end of the project to complete any remaining functionality, testing, and the all-important documentation, the functionality that is achieved by the second submission deadline affects the highest rating achievable for the *Functionality* category (refer to Table 1). Functionality will only count as being achieved if it has been decently implemented.

In order to promote the writing of tests concurrently with the writing of code there is a requirement with regard to the amount of testing that has been achieved by the second submission. Specifically, *basic movement tests must be provided for all moving game objects* that have been implemented by this time. A five percentage-point penalty will apply if this is not achieved.

The first two submissions do not require any reports. Release notes need to be provided, and the commit containing the release's source code, test code and resources needs to be tagged (refer to Table 2). These submissions must meet the requirements given in Section 7. Non-compliant submissions will be penalized (see Section 8.3.1).

Functionality present in second submission	Highest rating achievable for *Functionality*
Less than basic functionality	Acceptable
Basic functionality	Good
Basic functionality plus one major feature	Excellent

Table 1: The highest *possible* rating that can be achieved for the *Functionality* category is determined by the functionality present in the second submission. So, for example, if you have implemented basic functionality by the second submission then you are eligible for a mark of Good provided you meet the criteria given in marking rubric by the final submission.

Deliverable	First & Second Submission	Final Submission
Tagged commit published as a GitHub release	✓	✓
Accompanying release notes	✓	✓
Project report		✓
2 × Academic Integrity declaration forms		✓
Signed work contribution declaration document		✓
Online work contribution declaration form completed		✓
Technical reference manual generated by doxygen		✓

Table 2: Deliverables required for each submission. The tagged commit for each release will be built using CMake to produce a game executable, a test executable, and documentation. The game executable must contain a splash screen with game instructions.

6.2 Final Submission Deliverables

Each project group (a group of two) must submit the deliverables listed in Table 2 and comply with the requirements in Section 7. In addition to the deliverables required in the earlier submissions, the final submission must include:

- A *Project Report* (limited to 8 single-column pages from start to finish) presenting the problem; the domain model; a discussion on how the domain model is translated into a design, including both static class structure and dynamic run-time behaviour; and a critique of both the final functionality achieved and the object-oriented design. *Do not give an abstract, background section or literature review for this report*.
- A digitally completed and signed *Academic Integrity Declaration* form *from each student*. If this form is not signed and submitted then you will receive a zero for the project.
- A single page *Work Contribution Declaration* signed by both project partners stating which partner did which parts of the project and specifying each partner's discretionary mark which will be added to the project mark (see Section 8.2). If no discretionary marks are specified, or if the form is not signed by both partners, then these marks will be forfeited.
- In addition, a Google form must be completed by each group member stating their contribution percentage. The value given on this form must match that on the *Work Contribution Declaration*. If the value does not match, or the Google form is not completed, then 1 percentage point will be deducted from the percentage stated on the *Work Contribution Declaration*.
- A low-level *Technical Reference Manual* explaining the source code to other programmers who might wish to understand and modify it. This manual will be automatically generated using CMake and Doxygen.

7 Use of GitHub

7.1 The Project Repo

Your project repo must be modelled on the demo Pong project and contain the following items located in the correct directories:

1. All the *source code* for the game in a directory called: `game-source-code`.
2. All the *test source code* in a directory called: `test-source-code`.
3. All the game resources (image files, font files, text files, audio files, etc.) in a directory called: `resources`.

In the demo project, you have been provided with a `CMakeLists.txt` file. You *must not modify this file*. We will use CMake along with this `CMakeLists.txt` file to build your executables and documentation.

Do not include in your repo:

- The source code for any of the libraries being used.
- Files that are produced as a by-product of the build but are not necessary for running the game. Such files include project files produced by the VS Code, object code files, and so on.
- Any executables or library binaries.

- The project report or any declaration documents.

7.2 Git/GitHub Workflows

Suggested workflows for the project are given in the submissions guide.

7.3 Project Releases

There are three aspects to a published project release on GitHub:

1. A tagged commit in the project repo forming the release,
2. A release title and release notes, and
3. A zip file containing the documentation (for the final release only).

These aspects are described in more detail below. Refer to the submissions guide for further instructions on tagging a release.

7.3.1 Release Tags

A commit on the *master/main* branch needs to be tagged for each release. The tags for the releases are to be named exactly as follows: `v1.0` (for the first submission), `v2.0` (for the second submission) and `v3.0` (for the final submission).

You can try out this process by creating a test release before any submission is due. For example, you could publish a test release with the tag `v0.1`.

7.3.2 Release Notes

Release notes need to be published on GitHub for each release describing, at a minimum, *added*, *changed* and *removed* functionality of the game which has been released. The release notes need to follow the keep a changelog format.

7.3.3 Release Assets - Required for the Final Release

Upload a *zip file* containing the following to the *Assets* section of the GitHub release:

1. A *PDF* of the project report.
2. *PDFs* of all the declaration forms.

The zip file should be named using your student numbers as follows:
`<student number 1>-<student number 2>.zip`

8 Assessment

8.1 The Marking Rubric

The rubric, which forms part of this project brief, indicates how the project will be assessed. Each category is rated from *Unacceptable* to *Excellent*. Each of these ratings corresponds to a

Rating	Mark
Unacceptable	0
Poor	20
Acceptable	55
Good	70
Excellent	95

Table 3: Ratings and associated marks

particular mark (shown below). The overall mark is determined by averaging the category marks.

If any category receives a score of *Unacceptable* then both students' overall marks are capped at 40%. Note, however, that the overall mark can be lower than 40%.

8.2 Teamwork Assessment

The ability to work well with others is fundamental to engineering, and to software engineering, in particular. This is why teamwork is explicitly one of the outcomes of this project. You may not work on your own and project partner changes will only be considered when a student de-registers or there is clear evidence that one member of the group is not contributing. You are required to act professionally and support each other in achieving the goals of the project. It is a good idea to state your expectations from each other before the project begins and to discuss how you will handle possible conflicts that could arise.

Both group members are expected to contribute fairly equally to the project. Each group of two is allocated ten percentage points in discretionary marks. It is up to the group to determine how to divide this. The division must be in terms of *whole numbers*; decimal numbers will be truncated. The discretionary marks may be evenly split (5% and 5%) if it is felt that both members contributed equally to the project. If this is not the case, the group member who has made a greater contribution can be acknowledged by granting them a larger share of the marks. The discretionary percentage points for each group member are added to the overall mark to determine that member's final project mark. In order for the discretionary marks to be granted both group members have to agree on how the percentage points are apportioned (refer to Section 6.2).

In order to account for gross differences in contributions to the project, the following rule applies. The percentage of source code lines (across both game and test code, and including comments and modified lines) contributed by each group member and present in a release will be determined using: `git summary --line *.cpp *.hpp *.c *.h`. Note, this tool is part of git-extras which is separate from Git and needs to be installed.

A penalty will be applied for *each* of the releases `v1.0` and `v2.0` in which the percentage contribution of one group member is less than 35%. The group member with the less than 35% contribution will lose 5 percentage points from their final mark, in each case. If the percentage contribution of one group member is less than 35% for the final release (`v3.0`) then that group member's overall mark for the project will be capped at 40%. Note that under no circumstances may one group member commit work on behalf of the other group member.

8.3 Penalties and Bonus

8.3.1 Non-Compliant Submissions

Five percentage points will be deducted from the final project mark for each submission not meeting the submission requirements. This is independent of the evaluation of the deliverables submitted. A submission is non-compliant in the following circumstances:

- The release is not correctly published on GitHub (Section 7.3).
- The executables, built by CMake running on Windows, cannot be run *for any reason*. For the final release, any broken executable will result in a rating of *Unacceptable* on the rows of the rubric where that executable is assessed. Using CMake with the `CMakeLists.txt` file that is provided with the demo game will ensure that your game and test executables include all of their required dependencies.
- One or more commits are not linked to a group member's GitHub profile. The penalty will only be applied to the group member with unlinked commits.
- Releases are tagged on different branches.
- Any commit, on the *master* or *main* branch, leading up to a release (including the release commit itself) has more than 100 *inserted* lines of source and test code when compared to the previous commit on the *master/main* branch. The commits that will be analysed are those made directly to *main/master* and those merged into *main/master*. Merge commits, themselves, will be excluded from this analysis. The number of *insertions* between two commits can be seen using:
  `git diff --shortstat <earlier-commit-hash> <later-commit-hash>`.
  The penalty will only be applied to the group member with the too large commit.
- A commit that is on a branch leading to a tagged release is not an atomic commit. In other words, the commit does not build. The commits leading up to a release will be randomly sampled and built using CMake. If the build fails then the penalty will apply to the committer.
- The release assets file is not a zip file but instead is some other archive format.
- The game's source code is duplicated (partly or completely) within the `test-source-code` directory.
- The splash screen is missing, incorrect, or incomplete.
- For the second submission only: basic movement tests are not provided for each game object that has been implemented.

Note, the non-compliance penalty for any given submission is five percentage points even if there are multiple issues with the submission.

8.3.2 Late Submissions

Five percentage points will be deducted from the final mark for each interim submission that is submitted late. If the second submission is not received by 16:30 on the day of the deadline then, in addition to a five percentage point penalty, it will be taken that no submission was attempted and the highest achievable rating for the *Functionality* category will be *Acceptable*. For the two interim submissions, if there is a compliance issue as well as a late submission, the penalty will still be capped at five percentage points.

The School's Late Submission Penalty Policy will be strictly applied to the final deadline. The policy must be read and understood by the student. For the final submission, if there is a compliance issue as well as a late submission, then the two penalties will be added.

8.3.3 Early Hand-In

Groups who submit *all* of their project deliverables by 17:00 on the day before the final deadline, will enjoy a bonus of five percentage points, and a relaxed evening.

9 Plagiarism

Plagiarism detection software will be used to compare project submissions to one another and to code available on the internet. All instances of plagiarism will be severely dealt with. No two groups may have identical or overly similar deliverables.

10 Final Thoughts

It is critical that you apply a balanced effort to all aspects of the project and that you do not fall into the common trap of over-focusing on coding, and functionality, and neglecting other important activities such as design, critical analysis, testing and documentation. Remember that implementing additional functionality will in turn require more effort in all other areas of the project. It is invariably better to produce a final product that is acceptable in all respects, rather than a product that is excellent in only one respect and poor in all others.

Good luck and have fun!

References

[1] Wikipedia. "Dig-Dug." https://en.wikipedia.org/wiki/Dig_Dug, Last accessed: August 2025.

University of the Witwatersrand, Johannesburg
School of Electrical and Information Engineering
Software Development II

PROJECT ASSESSMENT FORM V0.43

	Unacceptable	Poor	Acceptable	Good	Excellent
Problem Understanding, Solution and Evaluation: project report	extremely flawed problem understanding/specification/ domain model, key functionality and/or design choices not explained at all, presentation of solution does not match implementation	poor problem understanding/ specification/domain model, key functionality and/or design choices hardly explained, fundamental misunderstandings of the implementation, class responsibilities inadequately described, blind acceptance of clearly defective functionality	adequate problem understanding/specification and domain model, class responsibilities described, some description of dynamic behaviour, minimal/flawed critique of the final solution in terms of functionality and/or design	good understanding/specification and domain model, class responsibilities well described, dynamic behaviour correctly depicted with sequence diagrams, reasonable critique of the final solution in terms of both functionality and design, good use of diagrams to communicate concepts	astute understanding, specification and domain model, class responsibilities are well described and dynamic behaviour well illustrated with sequence diagrams, excellent critique of the final solution in terms of both functionality and design, consideration of the broader problem domain, excellent use of diagrams to communicate concepts
C++ Design and Implementation: source code	raylib-cpp not used, implementation violates constraints, not object-oriented, no user-defined classes, GitHub not used for version control and publishing releases	poorly chosen abstractions or many key abstractions missing, poorly designed class interfaces, inappropriate relationships between classes, patent violation of fundamental principles such as DRY, implementation more like C than C++	abstractions generally have acceptable/appropriate behaviour but some key ones may be missing, acceptable class interface design but implementation may not be well hidden, mostly acceptable class relationships, modern, idiomatic C++ mostly used	*2 out of 4*: 1) well-modelled abstractions at all levels of granularity with good interfaces which hide information 2) clean separation of presentation and logic layers 3) small classes and no long functions 4) good use of role modelling. No clearly wrong design decisions, modern, idiomatic C++ used + good technical reference manual	*3 out of 4*: 1) well-modelled abstractions at suitable levels of granularity, and at all layers, with good interfaces which hide information 2) clean separation of presentation and logic layers 3) small, cohesive classes and no long functions 4) good use of role modelling. No clearly wrong design decisions, modern, idiomatic C++ used + good technical reference manual
Functionality: game executable	executable does not run, game functionality is at the level of the first submission or less	game runs but has major functional flaws	all basic functionality working acceptably	all basic functionality working plus 3 minor features OR 1 major feature and 1 minor feature	all basic functionality working plus 2 major features and 3 minor features
Automated Testing: test executable and source code	executable does not run, no genuine attempt at unit testing, doctest framework not used	insufficient testing - not meeting the requirement for *Acceptable*; or very limited testing due to insufficient functionality	automated test coverage of game logic is adequate and includes basic movement and collision testing for *all* game objects, some important game logic is not tested	automated test coverage of game logic includes all classes/functions responsible for the movement and collision of game objects, fair test coverage of features implemented, testing is thorough and test code is of good quality (good test names, easily understandable code, etc.)	distinguished from *Good* by one or more factors: comprehensive coverage of all functionality implemented; advanced use of testing framework, automated tests given for difficult-to-test functionality eg. involving randomness, gui interactions etc.
Technical Communication: project report	-20% report deviates significantly from the Blue Book	-5% report does not conform to the Blue Book, use of language, style and tone is poor, report structure is poor, report exceeds page limit	0%		

Notes:
All categories are equally weighted
If any category receives a rating of Unacceptable then both students' marks are capped at 40%

Bonus and Penalties:
Non-compliant submissions: first: -5; second: -5; final: -5
Early hand-in: +5; Late final submission: within first hour: -5; before 16h30: -15