CSSE220 - Dig Dug Project
Team Programming Assignment

You will write a game that is patterned off the 1980's Dig Dug game. You can find a description of the game, and much more information here:

https://en.wikipedia.org/wiki/Dig_Dug

You can also find an online playable version of the game here:

http://emulator.online/nes/dig-dug/

Table of Contents

Essential features of your program

Your graphics do not have to be fancy such as figures that animate or look like the original graphics. Actually, everything could just be represented by different colored rectangles/circles etc. You are graded on the functionality your program implements including:

- **A "hero" who moves, digs and inflates enemies**
- **To inflate the hero presses a button that lets a cord out. If the cord contacts a monster the monster stops moving. Then the hero can inflate the enemy with repeated button presses. After 3 or 4 inflates the monster dies.**
- **The classic game has 2 kinds of monsters, Pookas and Fygars. The monster's movement does not need to match the movement of the original game, but they should move in qualitatively different ways (i.e. one should not just be a faster or tougher version of the other – they should act visibly differently)**
- **Monsters should not do obviously stupid things such as getting stuck in corners of the board.**
- **Fygars should breathe fire occasionally, which should kill the player.**
- **Both kinds of monsters are constrained to the tunnels in their usual form, but they can change to a ghost form that can move through dirt. Once they are within a tunnel they should turn back to regular monsters.**
- **The game should have rocks that fall a few seconds after you dig the dirt under them. A falling rock should kill the player and/or the monsters that it touches.**
- **After 2 rocks are dropped per level, some fruit should appear in the center of the level. Picking up the fruit should give additional points.**
- ***Y*our game should load pre-created levels with planned configurations of the board and enemies. Different levels should have different numbers of monsters and different positions of tunnels. You do not have to exactly match the levels of the real game. A level should be representable by a text file. Such a file can be passed to a Level constructor method to create that level. A level file should include the starting locations of the hero, monsters, and power-ups. When the user selects "Play Game", the program should open the Level 1 file, and build the board layout based on what is in that file. Your levels do NOT need to scroll.**
- **Contact with the monsters kills the hero. When the hero dies, he and the monsters return to the start position, but the board stays in the current state. After a certain number of deaths, the player loses and has to restart the game from the beginning.**
- **Defeating all the enemies on the level should take you to the next level.**
- **The game should display the score. Digging and killing monsters should increase the score.**
- **Pressing the U key should cause the game to go up to the next level (i.e. from level 1 to level 2, level 2 to level 3); the D key takes you down to the previous level. These features are not in the sample game, but they will be very helpful for your (and your instructor's) testing of your game.**
- **Pause or restart the action by pressing the P key.**

Nice features to add

For this project we would like you to go beyond the minimum functionality and add some features that seem exciting and fun to you. If you accomplish only the "essential" features, you'll only get 85% of the functionality credit. To get to a full 100%, add some more features. If you implement a lot of features you can even get a little extra credit.

Additional features that you might add include

- **Images for the player, monsters, environment, power-ups**
- **Different kinds of weapons**
- **Even more qualitatively different kinds of enemies**
- **Different kinds of power-ups**
- **Save the game that is in progress, and load previously saved games**
- **High score list, where you can enter your initials after a successful game (maybe even that saves between different runs)**
- **Help screen that explains the keys (this is a minor one)**
- **Start screen with cool animations**
- **Animation of sprites that represent the characters**
- **Boss fight level where you must defeat a giant enemy**
- **Something creative that you want to add**

A major goal of this project

Your team should explore the various classes associated with Java Swing in order to find ways to do various things that you need. We hope this project will help you make the transition from just getting info about classes from the textbook and your instructor to also digging a lot of it out on your own. You may also want to research some general topics, for example, animation using Threads.

Reading and research may occupy a very significant portion of the time your team spends on this project.

Each team member should check out the ArcadeGameProject from the team's repository, and all subsequent work should be placed in your project folder and committed back to the repository.

Don't forget to minimize conflicts in source control by always updating before editing and before committing.

Parallel work

Between now and the end of the term, this project will occupy a lot of your programming time. But there will still be a few daily programming assignments along the way.

Development cycles

You will do this program in several short development cycles, most lasting three or four days; the last one only one day. Before the beginning of each cycle, you will list some features that describe what functionality should be present at the end of that cycle.

Milestones

Key points:

1. To get credit for the milestones, every student should have submitted code (we estimate at least 50 lines per person per milestone)

2. The code checked into your source control must work (i.e. should compile and run directly from source control with no special tricks)

Cycle 0: UML Class Diagram

You should go through the same kind of process that we used in the in-class Email exercise:

1. Brainstorm possible classes. (We would guess that you will come up with about 8-12 classes but more are certainly possible)

2. Assign responsibilities to classes; determine how classes need to collaborate in order to carry out those responsibilities, and what responsibilities those collaborating classes need to have. Will inheritance or interfaces help you to organize the responsibilities? Keep iterating this until all of the program's responsibilities have been assigned to classes.

3. Collect the information into a UML class diagram. Your diagram MUST be computer generated – use UMLet (easy drag and drop) or PlantUML (sort of a coding language which is what we use to generate diagrams for class). PlantUML has a Google Doc chrome extension which might be nice to use if you need to collaborate remotely.

Save your diagram as a PDF or JPG file, so it can be viewed without UMLet. Submit both your UMLLet data file and a PDF/JPG version in the eclipse folder you commit.

Begin implementing, commenting, and testing your code, cycle by cycle. We've included suggestions for what an appropriate amount of functionality for each cycle would be – but feel free to get ahead of us (especially if you've got a particularly cool extra feature planned). If you want to do features in a different order – get permission from your TA or professor.

Document your code as you go along.

Cycle 1: Levels

Minimum functionality:

- **Levels loading from files**
- **Dirt that the player digs through**
- **A hero that can move and shoot a pump, but the pump does not do anything**
- **Switching between loaded levels with U (Level 1 -> Level 2...) and D (Level 2 -> Level 1...)**

Cycle 2: Monsters

Minimum functionality:

- **One kind of monster with motion and which can kill players**
- **Monster can be inflated by pump**
- **Monster can turn into ghost form and move through walls**
- **Rocks that the player cannot move though**

Cycle 3: More

Minimum functionality:

- **Second kind of monster**
- **Rocks drop after a delay when dirt has been removed from under them**
- **Dropping rocks kill hero and monsters**
- **Fruit appears in center**
- **Points**
- **Advancing the next level on victory**
- **Everything else in the basic game**

Cycle 4: Extras!

Minimum functionality:

- **Whatever features you team wants to add!**

Commit your project often. Commit messages are important when working with a team, and especially include a message like END_OF_MILESTONE_*n* when you submit your working code for milestone *n*.

Code-in-progress

Your code should always run at the end of each milestone deadline. Your code should always be well designed and use good style. Be aware that your design and style will be evaluated at the end of the course.

Teamwork and grading

This assignment will be done by two-to-four-person teams. Our intention is not that you "divide and conquer" so much as that you have someone to talk with as you write and test this program. If you have not already done so, read this short article on Pair Programming and discuss it with your partners: http://en.wikipedia.org/wiki/Pair_programming . In particular, note what it says about who should be the driver if you are a "mismatched pair."

All code that you submit for this project should be understood by all team members. It is your responsibility to (a) Not submit anything without first discussing it with your partners, and (b) not let something your partners write go "over your head" without making a strong effort to understand it, including having your partners explain it to you of course.

If your team is having a problem with members not working together, please bring it up with your instructor ASAP. If you let us know at the end, there is little we can do to help.

It is possible that different team members will receive different scores for the project, if there is ample evidence that one person did not fully participate in the learning and the doing (or that one person "hijacked" the project by insisting on doing most of it without much help or understanding from the rest of the team). We reserve the right to give different grades. A peer evaluation survey at the end of the project will help us determine this. If the survey or our observations indicate that you do not understand, we may ask you to explain parts of your project code to us.

We will expect your evaluation of your team members at the end of the project to be detailed and specific. You should be writing it as you go through the project. Make notes of both positive things and suggestions for improvement. Then when it is time to submit your evaluation, you can mostly just paste what you have written into the Moodle survey.

Final Working Software

We'll grade the version of your software committed to your repository at the final-working-software deadline. Your code should be well-commented and should use appropriate class, method, field, and variable names.

Some comments on Subversion and team projects:

Commit your code often. And don't forget to update your code before editing and before committing. The chances of SVN conflicts grow exponentially with the number of team members, but they decrease with the number of lines of code in the project. The net result is that you'll have more trouble at the beginning of a project. For this reason it makes a lot of sense to program as a group or to carefully work on completely different classes in the beginning.

Style and Correctness

While implementing great features to your game is fun and important, it is also important that you apply the object-oriented design principles that you learned in this course to your design and coding. This 50 point section covers such things as code clarity, avoiding duplicated code, high cohesion and low coupling, and proper use of polymorphism and dynamic dispatch. Therefore, there will be significant deductions for code that, for example,

- **has a few large classes (like Main) with low cohesion,**
- **uses any static variables to avoid passing objects through parameters,**
- **uses type-predicated code. That is when one class uses if statements based on the type of another class (using instanceof, the name of a class, or even a special getType() function that returns a string or integer code).**
- **has high coupling (like a class that refers to most other classes),**
- **is uncommented,**
  or
- **contains any duplicated code whether within a single function, mostly identical functions within a class, or similar functions across different classes. Utility functions or inheritance should be used to remove this duplicated code.**

Presentation

Your team will give a 10 minute presentation on your project, which may be open to the Rose-Hulman community. Your goals for this presentation are:

- **Confidently and professionally describe your results.**
- **Demonstrate a sampling of the required and additional features that you've implemented.**
- **Show off bonus features that you've implemented.**
- **Describe the basic design of your system and discuss the amount of cohesion and coupling in your design.**

Every team member should play a significant role in the delivery of your presentation.

Keep in mind that all of us have implemented the same basic project, so you won't have to spend much time describing the basics of the project.

Grade components

15 points	**Initial UML diagram**
30 each	**Code functionality for Cycles 1, 2, and 3**
140 points	**Final program functionality and correctness**
50 points	**Style and efficiency**
25 points	**In-class presentation**
25 points	**Thoughtful team evaluation and reflection on the project (individual)**
??	**Additional features (extra credit)**

Disclaimer: This document may be revised in response to student questions/corrections. The latest version will be considered the authoritative one. If any changes significantly modify or clarify the project requirements, we will notify all students by email, to make sure that you read the new version of this document.