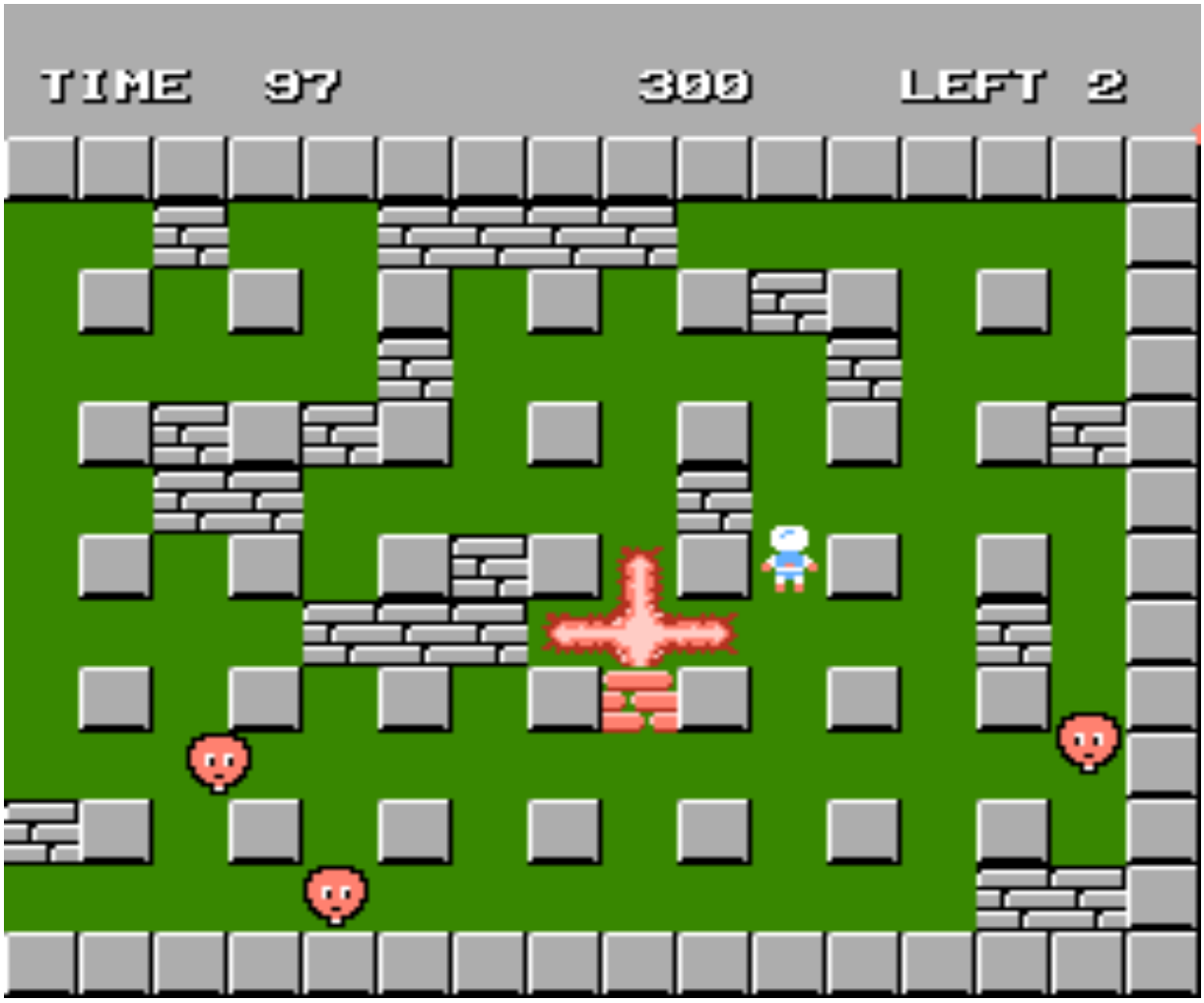
CSSE220 BomberMan programming assignment – Team Project



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

http://strategywiki.org/wiki/Bomberman

Also, the original NES instructions manual:

http://www.gamesdatabase.org/Media/SYSTEM/Nintendo_NES/manual/Formated/Bomberman_-_1987_-_Hudson_Soft.pdf

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

http://game-oldies.com/play-online/bomberman-nintendo-nes#

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" that moves and sets bombs
- There are 8 kinds of monsters, as detailed in the wikipedia page. The BomberMan hero should use the classic movement of the original BomberMan game. For the other monsters, you do not have to duplicate their movement style exactly but the monsters should do the same thing as in the classic game.
- Monsters should also not do obviously stupid stuff like get stuck in corners of the board.
- Some portions of the board can be destroyed by player bombs. Enemies, the brick walls and the hero all should die if in the path of an exploding bomb.
- *Unlike the classic BomberMan game*, your 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 initial items. 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.
- *Unlike the classic BomberMan game*, you game should have a least 3 different kinds of weapons. These weapons should be qualitatively different from each other (i.e. not just stronger or faster versions of the main weapon). For example, you could have a weapon that causes your bombs to explode on hit damaging nearby monsters, and even a moving bullet. Other choices are possible too, but the weapons should be DIFFERENT. Pressing the number keys should switch between weapons.
- 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.
- Pressing the U key should cause the game to go up to the next level; 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
- Even more qualitatively different weapons
- Even more qualitatively different kinds of enemies
- Even more qualitatively 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 (all due at the beginning of class, except as noted)

Key points:

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

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

3. The code should implement all the milestone requirements

(Day 22) – See schedule for date	Cycle 0 due
(Day 24) – See schedule for date	UML class diagram, Cycle 1 code, progress report, and feature list for Cycle 2 due
(Day 25) – See schedule for date	Cycle 2 code and progress report, feature list for Cycle 3
(Day 28) – See schedule for date	Cycle 3 code and progress report, feature list for Cycle 4
(Day 30) – See schedule for date	Final Code and documentation (see grade components below), Project Demo in class
See schedule for date (DUE BEFORE FINAL EXAM)	Team member evaluation survey (required if you want a grade for this project)

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.

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 Moodle assignment dropbox.

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
- Walls that the player cannot move through
- A hero that can move and fire 1 kind of bomb
- Switching between loaded levels with U and D

Cycle 2: Hero and Monster

Minimum functionality:

- Classic BomberMan monsters
- Killing the hero
- Killing the monsters
- Destroying the brick walls, but not the concrete walls

Cycle 3: More

Minimum functionality:

- All kinds of enemies
- All kinds of guns/bombs
- Power-ups
- Going between levels on victory
- Everything else in the basic game

Cycle 4: Extras!

Minimum functionality:

- Whatever features you team wants to add!

Commit your project often.

Status Reports, Code-in-progress and Team Evaluations

At the end of each development cycle, you will commit a text document to your project repository that lists the features then add the *actual time* you spent on each. Indicate either that you completed all of the features for the cycle or else list any features the team planned to complete but was not able to. Briefly state any complications that prevented you from completing the stories, for example, “We underestimated how hard it would be to implement mouse dragging.”

You can just modify the document named “Cycle *N* Status Report.txt”, where *N* is the cycle number just completed, in the Planning folder. Commit the file to your repository.

You should be using good process as you go. Thus, for each cycle, your code must run, have good style and complete documentation. It should also have no other warnings, although sometimes these are inevitable (like if you have declared a variable you will use in the next cycle).

Teamwork and grading

This assignment will be done by three-to-four-person teams. If the number of students in your section is not a multiple of four, there may be one or two teams of three students. 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.

This project should give you practice with the "short cycles and feature list" approach to software design and implementation.

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. No Eclipse warnings should remain for your final code according to our standard Eclipse preferences for CSSE 220.

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.

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.