CSSE220 Bomb Jack Team Project

Graphical user interface

Description automatically generated

You will write a game that is based on the Bomb Jack arcade game

**Bomb Jack Wikipedia Description**: <https://en.wikipedia.org/wiki/Bomb_Jack>

**A YouTube video of Bomb Jack being played**: <https://www.youtube.com/watch?v=fwXnoZprGXY>

[1. Essential features of your program 3](#_Toc84847842)

[2. Teams, Calendar and Initial Requirements 5](#_Toc84847843)

[2.1 Shared Calendar 5](#_Toc84847844)

[3. Milestones – General Requirements 5](#_Toc84847845)

[Initial Arcade Git Repo Setup (10 points) 6](#_Toc84847846)

[Milestone 0: UML Class Diagram & Shared Calendar Setup (50 points) 6](#_Toc84847847)

[Milestone 1: Levels and Hero Movement (50 points) 6](#_Toc84847848)

[Milestone 2: More Movement and Aliens (50 points) 7](#_Toc84847849)

[Milestone 3: Collisions (50 points) 7](#_Toc84847850)

[Milestone 4: Final Submission, i.e., Final Commit to Your Repo (50 points) 7](#_Toc84847851)

[Milestone 4: Final UML Diagram (50 points) 7](#_Toc84847852)

[Milestone 4: Extras! (50 points) 8](#_Toc84847853)

[4. Code-in-progress 8](#_Toc84847854)

[5. Teamwork (50 points) 8](#_Toc84847855)

[6. Style and Correctness (50 points) 9](#_Toc84847856)

[7. Presentation (50 points) 9](#_Toc84847857)

[8. Grade components 10](#_Toc84847858)

[Appendix A – Example UML Diagrams 11](#_Toc84847859)

# 1. Essential features of your program

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

* **Hero**:  
  A “hero” (Bomb Jack) who moves and flies, stands on platforms, falls, and collects objects, starts out with *X* number of lives, where you choose *X* > 1
* **Enemies**:   
  The classic version of this game has multiple levels; each level has at least 1 type of alien that can kill the *hero.* The alien’s form (shape) changes and kinds of movement changes from one level to another level. To begin with, make all aliens on a level be present when the play starts at that level. Later, you may want to add (as a special feature) having additional aliens appear on the level after play has already begun for that level.   
  On the *initial level* the aliens just move around with no real purpose, i.e., they are not actively tracking the hero. Beginning on the *second level*, at least some of the aliens must actively track and chase the hero.   
  A collision between an alien and the hero and can cause the hero to lose a life. Your game, when completed, needs at a minimum 2 types of aliens, the alien’s movement does not need to match the movement of the original game, but aliens should be able to fly, actively track, collide with the hero, and should never get stuck. The second type of alien might be able to shoot at the hero, better chase the hero (e.g., use better tracking algorithms), go faster, etc. Use your imagination.
* **Collisions and optional shooting**:  
  When the hero and an alien collide, the hero dies and loses a life only if the hero’s altitude is less than or equal to the alien’s altitude, otherwise, the alien is destroyed and the hero scores points. For example, if the hero’s location is (xh, yh) and the alien’s location is (xa, ya), then if yh ≤ ya, then the hero loses a life, otherwise the alien is destroyed  
  *If* you implement an alien shooting capability, and if the hero is hit by (i.e., collides with) an alien bullet, the hero dies and loses a life  
  *If* you implement a hero shooting capability, and if an alien is hit by the hero’s bullet, then the alien is destroyed and disappears from the screen and the hero scores points
* **Picking up objects**:  
  There are a fixed number of bombs per level, they do not move, and they are place at various locations on a level. These bombs are to be picked up (collected) by the hero. The hero collects a bomb by colliding with it. Each time a bomb is collected, the hero scores points.   
    
  In the actual game, a bomb’s fuse can become lit and then explode before the hero picks it up. Exploded bombs cannot be collected so no points can be scored. For our game, you *do not have to implement* exploding bombs. However, in later milestones, you might consider implementing this feature to earn incentive points.
* **Winning a level**:  
  If the hero collects all the bombs on a level, then that level has been won by the player controlling the hero. Your game should award additional *special bonus* points to the hero for winning a level, you can determine the formula for the special bonus points. After bonus points have been awarded, play resumes on a new level where that new level has its own configuration of bombs and aliens.
* **Game over:**When the hero loses all *X* lives (either by collisions with aliens, or by being shot), then the game is over
* **Configuring the levels:**Your game should load each level from a pre-created text-based data file that contains information concerning the location of platforms, aliens, bombs, and the hero. **Each level should have its own unique configuration of platforms, aliens, bombs, and hero**. Once play starts on a level then aliens begin moving around on that level. You can construct these text-based data files by using a simple document editor. This type of file can be read into your app by a Level Constructor method in order to create that level. When the user selects “Play Game”, your app should open and read in the Level 1 file and build the board layout based on what is contained in that file. Your levels do NOT need to be bigger than one full screen.
* **Keeping score:**At a minimum, the game should have a score that’s displayed and the number of lives the hero currently has
* **Special keystrokes:**You must implement 2 special keystrokes: pressing the *U* key causes the game to go up to the next level; the *D* key takes the game down to the previous level. These features are not in the sample game, but they will be very helpful for you, the TA, and your instructor when testing and grading are being performed.
* **Nice features to add:**  
  One of the many parts to the project is extra functionality, i.e., extra features, that you add to your game. If you implement a lot of features, then you can earn a little extra credit that could help improve your overall grade in the class.  
    
  Here are some additional features that you might add include (but are not limited to)
* Implement alien and/or hero shooting capabilities
* Images for the player, aliens, environment
* The original game has bombs whose fuses can become lit and then explode, this can be an extra feature
* The original game periodically has a circular bouncing “P” appear, and when collected by the hero, it will turn all the enemies into bonus coins for a short period during which Jack may collect them
* Have power-ups
* Different kinds of weapons
* More than one player at the same time
* Even more qualitatively different kinds of enemies
* Save the game that is in progress, and load previously saved games
* Maintain a 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) - but you should consider doing this so that when the instructor or grader plays your game, they will know what each key does
* Start screen with cool animations
* Animation of sprites that represent the hero and/or aliens
* Boss fight level where you must defeat a giant enemy/alien
* Something creative that you want to add

# 2. Teams, Calendar, and Initial Requirements

Teams have already been selected based on a survey previously sent out. The teams and the TAs assigned to each team can be found in the Final Project Teams and TA Assignments spreadsheet. When you need help with your project, your point of contact is your assigned TA. You will receive prompt feedback from each Milestone, including your initial UML Diagram (Milestone 0) from your assigned TA. All materials turned in must be submitted so that your assigned TA may access the materials. For further details, you should speak with your assigned TA.

## 2.1 Shared Calendar

A very important part of your project is creating a shared Google calendar. When you create this calendar, it should be accessible and editable by all teammates, your assigned TA, and instructor. Use the Google calendar share feature to accomplish sharing. he TA nor the instructor need to be able to edit the calendar, but they should be able to always view it.

In this calendar, you should include all of the following:

* All team meetings, both impromptu and planned meetings
* All occasions spent coding on the project; please indicate the person or people who worked on it at the time, i.e., whether an individual or the team worked on the project at the specified time
* Any occasion spent working on the project in any way, including who worked, when and how long

Once you discover who your teammate is, you must meet to discuss the project and create the calendar as soon as possible. You must discuss the plan for the project, when and how often you plan to meet and then enter this information into the shared calendar. You must start on the UML diagram right away and then move to the coding portion. The sooner you get started the better.

# 3. Milestones – General Requirements

1. Every team must commit code to their project’s repo – we estimate at least 50 lines per person per milestone, but it could be more
2. The submitted project (committed to your repo) must build and run with no special interventions by the TA or an instructor – if your project does not build and/or run, then zero points will be earned  
   *There are occasions where the program seems to work on your machine but the program does not work for the grader or an instructor, if this happens, your project will be graded based on how it runs on your machine, however, we must get this problem fixed so that it runs on the TA and instructor’s machine.*
3. For intermediate commits (i.e., between Milestones):  
   You must include a *commit message* indicating which team member or members worked on the chunk of code that is being committed and include a short description of what was worked on.  
   For example: “Rey Skywalker, worked on detecting collisions between the hero and aliens”
4. For a Milestone Commit  
   You must commit to your team’s repo with a commit message that says   
   “*This is the Milestone X commit and is ready for grading*”  
   Replace X in this message with *1* for Milestone 1, *2* for Milestone 2, etc.  
   Do not fail to do this *Milestone* *Commit*

## Initial Arcade Git Repo Setup (10 points)

*To earn 100% for this*

* *Your repo must be setup and accessible by all Team members, your assigned TA, and your instructor*

## Milestone 0: UML Class Diagram & Shared Calendar Setup (50 points)

*To earn 100% for M0*

* *You must complete Steps 1 – 5 (below)*
* *Your UML diagram should look at least somewhat like the provided examples – see Appendix A*

**Step 1 – (10%) Set up your shared calendar**

* See Section 2.1 (above) for instructions

**Step 2 – Brainstorm possible classes**

* We suggest that you will come up with about 6 to 10 classes, but more are certainly possible

**Step 3 – (60%) Assign responsibilities to classes**

* Determine how classes need to collaborate 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 through this step until all the program’s responsibilities have been assigned to classes.
* Reference Appendix A examples

**Step 4 – (30%) Construct a UML class diagram based on Steps 1 and 2**

* Your diagram MUST be computer generated
* Use UMLet (easy drag and drop) or PlantUML
* Save your diagram as a PDF, JPG, PNG, or SVG file

**Step 5 – Upload your UML class diagram to the Milestone 0 Moodle assignment**

* If you do not do this, then you earn zero percent for M0

**Step 6 – Begin implementation**

* Create your classes, commenting and testing your code, milestone by milestone.
* You are always free to work ahead, e.g., if you finish Milestone 1, you can immediately move on to working on Milestone 2
* Document your code as you go along – use JavaDoc comments as well as non-JavaDoc comments

## Milestone 1: Levels and Hero Movement (50 points)

*Functionality required to earn a 100% for M1*

* *At the least, M1 Items 1 - 4 (listed below)*
* *Successful completion of all M1 Items (1 through 7) at M1 grading time will be noted   
  This successful completion has potential to benefit your overall grade for the project*

1. (Failure to do this step is −100% for M1) Follow the instructions to rename your project, e.g., if your team’s name is “A\_039”, rename from *ArcadeGameGit-00* to *ArcadeGameGit-A\_039*
2. (30% toward M1) The initial game level (Level 1) must load from a file and be displayed by the app
3. (35% toward M1) Left and right movement by the hero controlled by the user using the left and right arrow keys
4. (35% toward M1) Up (flying) movement by the hero controlled by the user using the up-arrow key
   * No automatic movement by your software required for this milestone, this includes falling
   * No *diagonal* movement by hero is required for this milestone, this movement combines left/right movement *simultaneously with* up movement
5. (5% toward M2) Two different levels (Level 1 and Level 2) can be loaded from their respective files
6. (5% toward M2) Switching between levels by pressing the “U” (up) and “D” (down) keys on the keyboard

## Milestone 2: More Movement and Aliens (50 points)

*Functionality required to earn a 100% for M2*

* *M1 Items 1 – 5 (above)*
* *At the least, M2 Items 1, 2, and 3 (below)*
* *Successful completion of all M2 Items (1 through 3) at M2 grading time will be noted   
  This successful completion has potential to benefit your overall grade for the project*

1. (30% toward M2) Automatic movement/animation (things moving without keys pressed)
   * + The hero should fall when not on a platform (stops falling when it lands on a platform)
     + The fall should be gradual, not instantaneous
     + This movement/animation occurs without key presses
2. (30% toward M2) Two kinds of aliens that move in a reasonable way, tracking of the hero by an alien not required
3. (30% toward M2) Diagonal movement by hero
   * Flying diagonal movement – up arrow key in combination with left/right arrow key
   * Falling diagonal movement – left/right arrow key while falling
4. Begin planning for how your implementation will handle basic collisions (e.g., alien runs into hero and kills hero) but this will not be checked until the next milestone

## Milestone 3: Collisions (50 points)

*Functionality required to earn 100% for M3*

* *M2 Items 1 – 4 (above)*
* *At the least, M3 Items 1, 2, and 3 (below)*
* *Successful completion of all M3 Items at M3 grading time will be noted   
  This successful completion has potential to benefit your overall grade for the project*

1. (45% toward M3) Aliens colliding with hero – which means collisions must be handled
2. (45% toward M3) Collecting a bomb by the hero when with a bomb
3. (10% toward M3) Hero loses game when all lives are lost
4. (5% toward M4) Winning a level when hero collects all bombs

## Milestone 4: Final Submission, i.e., Final Commit to Your Repo (50 points)

*Functionality required to earn 100% for the M4-Final Submission*

* *M3 Items 1 – 4 (above)*
* *Since this is your final commit, the game and all features should be fully functional. Any functionality or features not finished at this time will not be considered for grading, anything added after the final commit will not earn credit*

## Milestone 4: Final UML Diagram (50 points)

*To earn 100% credit for M4-Final UML Diagram*

* *It is best if you make changes to your initial UML diagram as you make progress on your game so that your diagram is always up-to-date. If you do this, you only need to turn in your diagram as it is after all changes have been made.*
* *You must redo Steps 3 and 4 from Milestone 0 but this time your UML diagram must reflect the actual code that you submitted for Milestone 4*
* *There will be a place in Moodle to turn in your UML diagram, but it must be included with your code in the git repo. Please follow any additional instructions assigned to your team by your TA with respect to your final submission of your code and the final UML diagram.*

## Milestone 4: Extras! (50 points)

*Functionality required to earn 100% for M4-Extras*

* You must discuss with your assigned TA the extra features that you are considering for your project. Since each feature is different, therefore each having its own degree of difficulty, there is no predetermined set of features that we can prescribe that will cause your team to earn 100% credit for the Extras.   
  Rather, it is a “reasonable” amount of extra functionality that makes the game have your own look and feel. Once you discuss this with your assigned TA and together come up with your set of Extras, then if you go above and beyond this agreed upon set of Extras, these “above and beyond” features could help make up for any missed parts in previous milestones, or if you do extremely well in the project, this could aid in improving your overall grade for the course.
* See Section Nice Features to Add (near the top of this document)

# 4. Code-in-progress

Your implementation should always build and run for each milestone submission. 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.

# 5. Teamwork (50 points)

This assignment will be done by two-person teams, and in some cases a 1-person team. 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>. Note what the article 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 teamwork problems, e.g., have problems dividing up and/or doing the work, please bring it up with your *instructor* ASAP. We have some experience at finding ways to help team members work better together. 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.

# 6. Style and Correctness (50 points)

While implementing great features to your game is fun and important, it is also (arguably more) important that you apply the object-oriented design principles that you learned in this course to your design and coding. This requirement includes *code clarity*, *avoiding duplicated code*, *high cohesion*, *low coupling*, and *proper use of polymorphism* and *dynamic dispatch*.

Therefore, there will be significant deductions for code that has any of the following “no no’s”.

**Code and Design No-no’s:**

* Having just a few large classes (like Main) with low cohesion
* Use of static variables to avoid passing objects through parameters
* Use of 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)
* Having high coupling (like a class that refers to most other classes)
* Classes that are undocumented. Your classes must be documented with JavaDoc comments and with plain Java comments (where applicable)
* Classes containing duplicated code whether within a single function (commonly this occurs when students have blocks of mostly identical code for handling movement along the different cardinal directions), mostly identical functions within a class, or similar functions across different classes. Use procedural abstraction to create helper operations, or use inheritance to remove this duplicated code

# 7. Presentation (50 points)

Your team will give an approximately 7–8-minute presentation on your project (your instructor will give you specifics), 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 other teams have implemented the same basic project, so do not spend much time describing the basics of the project.

# 8. Grade components

1. Initial Arcade Git Repo Setup (10 points)
2. Milestone 0: UML Class Diagram & Shared Calendar Setup (50 points)
3. Milestone 1: Levels and Hero Movement (50 points)
4. Milestone 2: More Movement and Aliens (50 points)
5. Milestone 3: Collisions (50 points)
6. Milestone 4: Final Submission (50 points)
7. Milestone 4: Final UML Diagram (50 points)
8. Milestone 4: Extras! (50 points)
9. Teamwork (including your team/team member evaluations) (50 points)
10. Style and Correctness (50 points)
11. Presentation (50 points)
12. Extras Meriting Extra Credit (+ points)

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.

# Appendix A – Example UML Diagrams

Below are two UML diagrams to give you an idea of what you are expected to submit for Milestone 0. But of course, your diagrams will *not* be blurred like these example diagrams.

Diagram

Description automatically generated

Diagram

Description automatically generated