CSSE220 JetPac Team Project

A screenshot of a computer screen

Description automatically generated

You will write a game that is based on the JetPac arcade game

**JetPac Description**: <https://en.wikipedia.org/wiki/Jetpac>

**1 of many playable online versions**: <http://www.arcadedivision.com/classicgame49/shooting/jetpac.html>

**A YouTube video of JetPac being played**: <https://www.youtube.com/watch?v=9Hy29KoBnGY>

Table of Contents

[1. Essential features of your program 3](#_Toc53577394)

[Nice features to add 4](#_Toc53577395)

[Additional features that you might add include 4](#_Toc53577396)

[2. Milestones – General Requirements 4](#_Toc53577397)

[Milestone 0: UML Class Diagram (Pass/No-Pass) 4](#_Toc53577398)

[Milestone 1: Levels and Hero Movement (Pass/No-Pass) 5](#_Toc53577399)

[Milestone 2: More Movement and Aliens (Pass/No-Pass) 5](#_Toc53577400)

[Milestone 3: Collisions (Pass/No-Pass) 6](#_Toc53577401)

[Milestone 4: Final Submission (Pass/No-Pass) 6](#_Toc53577402)

[Milestone 4: Final UML Diagram (Pass/No-Pass) 6](#_Toc53577403)

[Milestone 4: Extras! (Pass/No-Pass) 6](#_Toc53577404)

[3. Code-in-progress 7](#_Toc53577405)

[4. Teamwork (Pass/No-Pass) 7](#_Toc53577406)

[5. Style and Correctness (Pass/No-Pass) 7](#_Toc53577407)

[6. Presentation (Pass/No-Pass) 8](#_Toc53577408)

[7. Grade components 8](#_Toc53577409)

[Appendix A – Example UML Diagrams 9](#_Toc53577410)

# 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. Actually, everything could just be represented by different colored rectangles/circles, etc. You are graded on the functionality your program implements including:

* **Hero**:  
  A “hero” who moves and flies, stands on platforms, falls, shoots, collects objects, and drops objects  
  The hero starts out with X number of lives, where you choose X > 1
* **Aliens**:   
  The classic version of this game has multiple levels; each level has 1 type of alien that defends its planet against the *hero.* The alien’s form (shape) and kinds of movement changes from level to level. On some levels the alien defends its planet by shooting at the hero on other levels the alien just collides with the hero. 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, try to attack and should not get stuck. The second type of alien must be able to shoot at the hero.
* **Collisions and shooting**:  
  When the hero and an alien collide, the hero dies and loses a life  
  When the hero is hit by a bullet, the hero dies and loses a life  
  When the hero loses all X lives, then the game is over  
  When an alien is hit by a bullet shot by the hero, then the alien dies and disappears from the screen and the hero scores points
* **Picking up and dropping objects**:  
  There are three types of objects to pick up: *rocket pieces*, *fuel objects*, and *valuable resource objects*  
  In the first level, the hero must first pick up rocket pieces and then drop them to assemble a spaceship.  
  Once the spaceship is assembled, then the hero must pick up fuel objects and drop them onto the space ship in order to fill up the space ship’s fuel tank. The hero scores points when picking up a fuel object.  
  When the hero picks up valuable resource objects, the hero scores points. The hero does not have to drop the valuable resource objects.
* **Winning a level**:  
  When an assembled spaceship is filled with fuel, then the hero has won the level. This causes 2 things to happen, (1) the spaceship takes off for space to a new level and (2) a new level is displayed and play resumes on this new level
* Your game should load pre-created levels with planned configurations of the board which contains platforms and a spaceship. Once play starts on a level then aliens begin defending that level and fuel objects and valuable resource objects begin to fall from space to platforms or to the ground (think of the ground as just another platform). **Different levels should have different numbers of aliens and different configurations of platforms (this is not the case in classic JetPac)**. Each level should be represented by a text file so that you can construct these using a simple document editor. This type of file can be read by a Level Constructor Java method in order to create that level. A level file should include the starting locations of the hero, platforms, and spaceship (if already assembled), or rocket pieces if that level contains a disassembled spaceship. 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.
* The game should have a score that’s displayed.
* 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 you and your instructor's for testing of your game.

## 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. One of the many parts to the project as a pass/fail portion is the extra functionality, i.e., extra features, that you add to your game. If you implement a lot of features you can even earn a little extra credit that could help improve your overall grade in the class.

### Additional features that you might add include (but are not limited to)

* Images for the player, aliens, environment, power-ups
* Different kinds of weapons
* More than one player at the same time
* 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 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 the CatMe survey previously sent out. The teams and the TAs assigned to each team can be found in the [Final Project Teams and TA Assignments](https://docs.google.com/spreadsheets/d/1g49eUzY12z_AKtfV2aAp0HSF3u2N2RlzMsdmcqLJI8A/edit?usp=sharing) 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.

## Shared Calendar

A very important part to your project, one part that is pass/fail, is creating a shared Google calendar. When you create this calendar, it should be accessible and editable by all teammates. When the calendar is created, you should invite your assigned TA, as well as your course instructor as viewers to the calendar. The TA nor the instructor need to be able to edit the calendar, but they should be able to view it at all times.

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

* Any and 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 should try and meet to discuss the project and create the calendar as soon as possible. You should discuss the overall plan of the project, when and how often you plan to meet and then enter this information into the shared calendar. You should attempt to start on the UML diagram when possible and then move to the coding portion. The sooner you get started the better.

# 3. Milestones – General Requirements

1. Every student should have submitted code – we estimate at least 50 lines per person per milestone, but it could be more
2. The code submitted must build and run with no special interventions by the grader – if it does not build and/or run, it will not Pass *(there are occasions where the program seems to work on your machine but the program does not work for the grader, if this happens, we will grade your project based on how it runs on your machine)*
3. For an intermediate commit between Milestones, add a commit message that indicates who worked on each of the parts that are being committed
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.

## Milestone 0: UML Class Diagram (Pass/No-Pass)

*To earn a Pass for M0*

* *You must complete Steps 1 – 4 (below)*
* *Your UML diagram should look at least somewhat similar to the provided examples – see Appendix A*

**Step 1 – Brainstorm possible classes.**

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

**Step 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 through this step until all of the program's responsibilities have been assigned to classes.
* Reference Appendix A examples

**Step 3 – 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, or PNG file

**Step 4 – Upload your UML class diagram to Moodle**

**Step 5 – 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 (Pass/No-Pass)

*Functionality required to earn a Pass M1*

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

1. The initial game level (Level 1) must load from a file and be displayed by the app
2. Left and right movement by the hero controlled by the user using the left and right arrow keys
3. 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
4. Two different levels (Level 1 and Level 2) can be loaded from their respective files
5. Switching between levels by pressing the "U" (up) and "D" (down) keys on the keyboard

## Milestone 2: More Movement and Aliens (Pass/No-Pass)

*Functionality required to earn a Pass M2*

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

1. 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. Two kinds of aliens that move in a reasonable way
3. Hero shoots bullets
4. One type of alien that shoots bullets
5. 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

* 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 (Pass/No-Pass)

*Functionality required to earn a Pass M3*

* *M2 Items 1 – 5 (above)*
* *At the least, M3 Items 1, 2, and 3 (below)*
* *Successful completion of all M3 Items will be noted at M3 grading time  
  It has potential to benefit your overall grade for the project*

1. Aliens colliding with hero – which means collisions must be handled
2. Shooting collisions
   * Hero shooting aliens, aliens disappear
   * Aliens shooting hero, hero loses life
3. Picking up object collisions
   * Hero picking up valuable objects and fuel objects
4. Dropping objects
   * Hero drops fuel objects on spaceship
   * Collision of fuel object with spaceship increasing spaceship's fuel level
5. Hero loses game when all lives are lost
6. Winning a level when rocket ship is completely fueled and then lifts off

## Milestone 4: Final Submission (Pass/No-Pass)

*Functionality required to earn a Pass M4-Final Submission*

* *M3 Items 1 – 6 (above)*
* *Since this is your final submission, the game and all features should be fully functional. Any functionality or features not finished at this time will not be considered as part of the project when grading the pass/fail for each part.*

## Milestone 4: Final UML Diagram (Pass/No-Pass)

*To earn a Pass 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 should be included with your code in the git repo. Please follow the instructions of your assigned TA as to your final submission of your code and the final UML diagram.*

## Milestone 4: Extras! (Pass/No-Pass)

*Functionality required to earn a Pass M4-Extras*

* *You should discuss with your assigned TA the extra features you are considering for your project. Due to the fact that each feature is different, therefore having different degrees of difficulty, there is no set number of features that we are looking for to pass this portion. Rather, it is a “reasonable” amount of extra functionality that makes the game your own. Once you discuss this with your assigned TA, if you go above and beyond that with other extras, this could help make up for any missed parts in the project, or if you do extremely well in the project, this could aid in improving your overall grade in 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 (Pass/No-Pass)

This assignment will be done by two-to-three-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. 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.

# 6. Style and Correctness (Pass/No-Pass)

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 not documented with JavaDoc comments and plain Java comments
* Classes containing duplicated code whether within a single function (commonly this occurs when students have blocks of mostly identical code for handling 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 (Pass/No-Pass)

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 Repo Setup (including shared Google calendar and git repository) (Pass/No-Pass)
2. Milestone 0: UML Class Diagram (Pass/No-Pass)
3. Milestone 1: Levels and Hero Movement (Pass/No-Pass)
4. Milestone 2: More Movement and Aliens (Pass/No-Pass)
5. Milestone 3: Collisions (Pass/No-Pass)
6. Milestone 4: Final Submission (Pass/No-Pass)
7. Milestone 4: Final UML Diagram (Pass/No-Pass)
8. Milestone 4: Extras! (Pass/No-Pass)
9. Teamwork (including your team/team member evaluations) (Pass/No-Pass)
10. Style and Correctness (Pass/No-Pass)
11. Presentation (Pass/No-Pass)
12. Extras Meriting Extra Credit (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.

# 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