Game Development using Advanced Programming: Angry Birds

1 Important Instructions

1. You must stick to the deadlines in this project as per the schedule. **No request for rescheduling the demo will be entertained.** In case of any unavoidable circumstances, you have to take email approval well in advance.
2. You must have a **private** Git repository on **GitHub** for your project and every group member should frequently check in their code in this repository. **Making your repository public before the end of the deadline and any plagiarism that results out of it will be dealt with appropriate disciplinary action.**
3. **Usage of ChatGPT, Claude, Gemini, and other LLMs** will be treated as **plagiarism**.
4. C**opying code, whether in full or partial**, from any other course student other than your teammate, or from online sources **without inclusion of due credit** to the source in your README file, will be treated as **plagiarism**.
5. If you see any ambiguity or inconsistency in a question, please seek clarification from the TAs by **commenting under the GC post** of the Project Deadline only. All doubts will be resolved in the comments, so keep an eye out on the comments for any clarifications the TAs/TF might offer.

2 Introduction

Angry Birds is a popular puzzle video game. It has a simple story: the pigs stole the birds' eggs, so the birds try to get them back. In every level, the player uses a slingshot to launch the birds at the pigs that are standing on buildings made of blocks. Each level gives the player a line of birds; the bird sitting on the slingshot is launched when the player pulls the bird back. The bird behind it jumps up, and it is the next bird to be launched.

When a bird is launched, it can hit the buildings and make the blocks break. This can make the pigs fall and disappear. The bird can also hit the pig itself to destroy it. The player has to destroy all the pigs in the map to unlock another level.

As a part of your major project for CSE 201 (Advanced Programming), you will be required to develop the Angry Birds game using Java.

You will be working in teams of two. The team must remain the same across all deadlines for the project. A separate form/sheet will be floated for you to register your team.

3 Getting Started

We will be using `libGDX` for game development in Java. `libGDX` is a cross-platform Java game development framework based on OpenGL (ES) that works on Windows, Linux, macOS, Android, your browser and iOS. You should read the `libGDX` documentation here to get a good idea of how you will be developing the game. You can visit the official website here for download links.

You should also play the game yourself and get a good idea of how the game works before you start implementing it.

4 Gameplay and Implementation

Some of the rules for the gameplay are mentioned below. You will get to know all the rules after you play the game and read the rules associated with the game. In your implementation, you should follow all those rules. Kindly make sure that you are using all the OOPs principles taught throughout the course (inheritance, polymorphism, interfaces, etc.), including at least 2 design patterns (which would be covered in the coming weeks), along with following best coding practices (naming conventions, access modifiers for class, fields, comments, etc.) and JUnit tests.

4.1 Basic Features

- You must implement the basic gameplay, where you have a set of finite birds to shoot at a structure containing pigs.
- A bird is shot by dragging it from a catapult and aiming it at the structure/pigs. The bird should follow the trajectory based on the angle and speed of the bird.
- The structure may be made of blocks of different materials like wood, glass, steel, etc., which might break on a single hit or multiple hits based on the material used.
- The collapse of a block will lead to all blocks above it dealing one hit each.
- A pig deals a hit if a bird hits it or the block immediately below it collapses.
- Based on the size of the pig, it might die on a single hit or multiple hits.
- You win the level if all pigs are dead.
- You lose the level if you have exhausted all the birds available to you.
- Each bird might have a different speed and impact. For example, a bird might deal one hit to each pig/structure it hits or it might deal two hits.
- You must implement at least 3 types of birds, 3 types of pigs, and 3 types of materials.
- Once a bird deals hits, the next bird should sit on the catapult.
- Each level has a different set of birds and a different structure and a different set of pigs.
- You should implement at least 3 different levels.

4.2 Serialisation and Game Saves

You should also have a save game function, which saves the current state of the level, including structures collapsed, hits dealt to each pig, remaining birds etc. You should also be able to restore the game using a restore menu. This should be accomplished using serialisation.

4.3 Bonus

You can implement more features than mentioned in this doc for bonus points. These bonus points will be decided by the TA evaluating your project based on the quality of bonus features implemented. A maximum of **2%** might be avoided on your project for all bonus features.

4.3.1 Examples

- Implementing special features for birds (eg. Black Bird can explode, Blue Bird can split into three, etc.).
- Implementing a random level generator, leading to infinite gameplay.

And whatever is up to your creativity!

5 Deadlines

We will have three deadlines for the project, and at each stage, the deliverables for the deadline will be evaluated. All of them will have a contribution to the final marks for this project.

5.1 Deadline 1 (23rd September): UML Class and Use Case Diagram

As a part of this deadline, you should submit a detailed UML Class Diagram and a UML Use Case Diagram.

- The **UML Class Diagram** should include all the **basic classes and interfaces**. They should have the necessary **attributes** and **methods** mentioned as well. **Class relationships** should be correctly identified and shown with the **class multiplicities**.
- The **UML Use Case Diagram** should include **use cases** where the **user interacts with the game**, with meaningful demonstration of **relationships and multiplicities** between them.

5.1.1 Deliverables

Submit two PDFs on the Google Classroom: one for the UML Class Diagram and one for the UML Use Case Diagram.

5.1.2 Weightage

This deadline will have a weightage of **2.5%** out of the 20% for the entire project. This will be split as follows:

- **UML Class Diagram:** 1.5%
- **UML Use Case Diagram:** 1%
- **Total:** 2.5%

You will be judged on the quality of your diagrams, specifically the classes identified, correctness of identified relationships, correctness of multiplicities, feasibility of the use cases identified, and the design patterns you use.

5.2 Deadline 2 (21st October): Static GUI

As a part of this deadline, you should submit a game with a **static GUI**. A static GUI means that you should have all **the components** and **visual elements** of your game, as well as the **events** implemented, but you do not need to implement the event handlers. This includes the Home Page, the Menus, the Game Screen, the In-Game Components, and the Pause/Level End Screens. Your repository should also have a **README file** detailing the commands to be run to **set up, run and test your project**, as well as any online sources you referred to.

5.2.1 Deliverables

Submit a **ZIP file** containing your **entire repository**, with the **README file** mentioned above.

5.2.2 Weightage

This deadline will have a weightage of **2.5%** out of the 20% for the entire project. You will be judged on the presence of all relevant components and screens, the presence of events to jump from one screen to another, and the visual quality of your static GUI, code quality and adherence to coding conventions, and the presence of README file with correct commands to set up and run your project.

5.3 Deadline 3(25th November): Functioning Game

As a part of this deadline, you should **submit the completed game**. The game should have **all the features implemented**. The game should be **serialisable**, and you should be able to save the state of your game (current level, progress within current level including all attributes and components of the level, solved levels, etc.). You should also **record a demo video**, where you show the functioning of each and every component of your game. You also need to create appropriate **JUnit Tests** to verify the functioning of different methods within your game. Your repository should also have a **README file** detailing the commands to be run to **set up, run and test your project**, as well as any online sources you referred to.

5.3.1 Deliverables

Submit a **ZIP file** containing your **entire repository**, with the **README file**, the game **demo video**, **JUnit Tests** and a sample **saved game**.

5.3.2 Weightage

This deadline will have a weightage of **15%** out of the 20% for the entire project. You will be judged on the completeness of the game, the presence of OOPs concepts such as Inheritance, Polymorphism, Interfaces, etc., the presence and completeness of the demo video, the presence and completeness of serialisation to save the game, the presence of JUnit Testing, the usage of design patterns, code quality and adherence to coding conventions, and the presence of README file with correct commands to set up and run the project.