Cpt_S 487 Project Specification

1. **Project Name:** BH-STG: Bullet Hell Shooting Game & Level Interpreter

Final project is an important component in a software design class. You will complete the project in a team environment (each team will have max 5, min 4 students, with 6 or 3-person team per instructor's permission or recommendation). The project is to provide a platform on which you can exercise software design and decision making on software architecture.

For this semester, your team project will be to implement a bullet hell shooting game, and its level interpreter, inspired by an indie Japanese doujin game series: Touhou Project. A sample game play could be found here: https://www.youtube.com/watch?v=-tyPdlhMLFQ
The genre mostly originated from the arcade games, notable examples include

- Touhou Project (http://en.wikipedia.org/wiki/Touhou),
- Galaxian (http://en.wikipedia.org/wiki/Galaxian, https://www.youtube.com/watch?v=XhYVcwhSWjI)
- Raiden: (http://en.wikipedia.org/wiki/Raiden_%28video_game%29, https://www.youtube.com/watch?v=Xe_5zxXiQAs)

Bullet hell is also a key element in one of the hit indie games: Undertale.

The basic concept of such games is quite simple: the player controls a character/spaceship, which shoots bullets/lasers at enemies to advance the game. Meanwhile, the player has to move and dodge a "large" number of enemies and the projectiles they fire. These projectiles sometimes form certain patterns to increase the difficulty and/or aesthetics of the game. At the end of each stage (or sometimes in the middle), the player will fight a "boss" who is usually more powerful and have more elaborated attacks. Other mechanisms (bomb, scores, power ups, etc.) are often available in such games as well.

The end goal for the project is to design and implement such a stand-alone desktop game and its corresponding level interpreter. There are three major deliverables for this project throughout the semester. This document explains the scope of this project, what is required for each deliverable, and other necessary details. Note that some requirements of the deliverables are intentionally withheld temporarily in this document. This is to encourage you to really think long and hard about your design before coding, and so that you might experience the challenges when facing changing requirements. These additional details will be announced in class once the previous deliverable is due.

2. Overall Project Instructions

1. **All codes should be submitted to gitlab.eecs.wsu.edu**. Your team should collaborate via git for team coding as well. DO NOT make the following activities regular occurrences within your team: email/message your codes back and forth; using a flash drive to carry the code; executable only on one team member's computer and that computer only; other similar methods.
   You should make learning git one of your main goals for this course. Occasional offline "copy and pasting" is of course fine, but don't make that into a habit.
   For the same reason, I will not accept zip files of a solution/project submitted via email or other methods. Make sure your project in your gitlab repository is runnable (with necessary libraries installed) on a normal PC with Windows (10 is preferred) after a fresh clone, and include detailed Readme file with instructions on how to run the program.
2. **You should use C# .NET or Java as your programming language.** Both languages work well with what we are trying to cover in this course. In particular, if you are using C#, please start researching on

MonoGame as your underlying game engine; if you are using Java, do the same for **libgdx**. All team members should get familiar with your choice of framework as soon as possible, as your first deliverable due date will be right upon you. *Unity* is NOT allowed. Other frameworks while not forbidden, but are not recommended. Please contact me if you would like to use some other frameworks to build the game.

3. **Important:** Please note that this course is not about how to code a specific game (and definitely not about how to play the game). Therefore, we will NOT cover any specific engine-related contents AT ALL, which means you are responsible for using all resources you can find to learn about **MonoGame** and/or **LibGDX**, and use them effectively to complete your project assignments. I strongly encourage setting up out-of-class discussion channels, such as a discord channel, if you need help and/or want to share wisdoms while learning the frameworks.

4. **Crucial: Read carefully and ask questions!** In order to include as many details as possible, this specification is terribly verbose, which will likely lead to confusions and even contain unintentional mistakes. Therefore, please be sure you read the requirements very carefully for each deliverable, and post any questions you might to the instructor.

5. **Demos:** Throughout the semesters, I will try to schedule team meetings (via skype/Hangout or even AMS video conferencing for Pullman teams) with each team every few weeks to check up on your progress, answer all your questions, get your feedback and keep in touch. These meetings will be announced in class in advance, as well as sent out via Canvas Announcement. The dead week will be used as the project demo week. Each team shall schedule a final demo with me during the week with everybody attending. You will demo everything you have done to me and explain your contribution.

6. **Grades:** details are included below, but the overall distribution of your project grade is as follows:
   a) Deliverable 1: 15pts (+up to 5 extra pts)
   b) Deliverable 2: 45pts (+up to 5 extra pts)
   c) Deliverable 3: 40pts (+up to 10 extra pts)

   Therefore, the overall possible team grade is 100pts (+up to 20 extra pts). In addition, every team will be considered for one of the three awards: **1) Best Features 2) Most User-friendly 3) Best Design**. The winner team of each award will be awarded another 10 extra pts.

   For the individual grades, we are taking into considerations of individual performances. While we understand that every person has different strengths, you are expected to pull your own weight and not counting on your teammates to carry you through. At the end of the semester, you will be asked to submit a peer review form that evaluates each individual's performance in your team by percentage. These reviews will remain confidential with me. The average of your percentage will be used to calculate your individual grades.

   Out of the standard team points, I take 20% out as the Individual Participation grade. For instance, in a 4-person team, each member's individual grade would be:

   TeamGrade * 80% + TeamGrade * 20% * (AvgIndPercentage / 25%)

7. **Another word on teamwork:** Teamwork is a crucial skill for a software engineer, and I expect everyone taking this course to take this with utmost seriousness and a collaborative attitude. This

project is by no means an easy one, therefore if one member does not apply oneself, the whole team might be negatively impacted.

Still, I recognize that unexpected situations do occur. Here are my suggestions and policies regarding teamwork issues:

a) If you think you are falling behind too much, be proactive and seek help from your teammates, classmates, and me. If you still feel overwhelmed, please let your teammates and me know about it, and your decisions about the circumstance. I have had students in the past who simply stopped showing up, and refused to respond to any forms of communication attempts. It certainly did not sit well with the remaining teammates, even if they were able to pull through.
b) If you are on the other side of the problem, finding yourself (yourselves) always have to carry the team, do most (even all) of the work constantly, also be sure to reach out to me, and we will try to work things out.
c) As a team, you should set a regular weekly meeting time to keep each other on the same page. Of course, I want to remind you not to do all your coding *only* when you are *physically together*. That is why we are using git – and learning the importance of appropriate good design and task division.

I hope we don't have to use any of these policies throughout the semester. Nevertheless, they are here to guarantee that if you work hard on the project, you should not have to worry about being treated unfairly.

8. Project resources:
   a) There are plenty of videos on YouTube to show you how the original game is played. If you'd like, you may even play the game yourself to get a better feeling of the mechanisms.
   b) Be aware of copyrights. If you use other people's codes, images, music/sound, font, etc., be sure to credit them properly.

3. Project Deliverable 1: see Due Date on Canvas (15pts)

Requirements: You need to build a playable game that is similar to the game shown in this video available on Canvas: "487-Project-SampleGamePlay.mp4"
Here are some definitions that are used in this document, with reference to the video's timestamp.

- Player/Player Character
  The character/ that the player can control in the game. Appeared at the beginning at the bottom of the screen. The hitbox, which appears as a white dot in the center of the sprite when "slow speed" mode is activated (see below in required features), is much smaller than the size of the whole sprite.
- Regular Enemies
  Non-boss enemies. There are different variations of such enemies. They move and/or attack by firing bullets. In the video, there are mostly two kinds: smaller grunts in red/blue/green, and the larger butterfly enemies that fly from one side to the other.
- Mid/Final Boss
  The Mid Boss (not regular grunts) appeared at 00:48 mark, and the Final Boss appeared at 01:30 mark. The bosses have more specific movements, and their attacks are more elaborated and complicated.
- Regular Stage
  All gameplay outside of the mid and final boss fight.
- Mid Boss/Final Boss Fight
  Mid boss fight: from 00:48 to 01:15. Final boss fight: from 01:32 to the end of the video. The final boss fight has 4 stages, each of which features a different movement pattern and bullet-firing pattern.

I. The key features **required** are:

1. The player. **(2pts)**
   a) A player character should be able to respond to keyboard control and move in the corresponding 8 directions. Its movements are confined in a certain area on screen.
   b) A player also can switch between "normal speed" and "slow speed" modes. The latter allows the player to move slower and more accurately for dodging.

2. Enemies and Bosses **(3pts)**
   a) Four types of enemies, including: regular enemies A and B, a mid boss, and a final boss, should spawn, move, and eventually exit throughout the progress of the game.
   b) The enemies/bosses should be able to move and fire bullets on their own while they are on screen.

3. Project Vision **(10pts)**
   a) Submit a txt file to your git repository, detailing your own vision of the game you are going to implement by the end of the semester. You are not required to replicate exactly what is in the video, but here are the requirements for your end project:
      i. The entire game should last approximately 2-3 minutes, making up of 4 phases: regular play with grunts; mid boss attack; more grunts; and final boss attack.
      ii. At least four different types of enemies, as stated above. The mid boss and a final boss must have more elaborated attacks.
      iii. Specifically, the final boss must have two different stages of attack, mimicking the first stage and third stage of attacks in the video (01:36 – 02:22, and 03:07 – 03:52, respectively). This is required to be in your final demo.
      iv. Other more detailed requirements are included below in **Deliverable 2 & 3.**

b) In the txt file, present as much game related details as you can think of. Describe your plan of the game in a chronological sequence, for instance, "The first wave lasts 10 seconds, XXX happens", and descriptions of the bosses' firing patterns etc. Think of this as a "game script" that provide base line ideas that your code will try to achieve.
c) Also include other information such as how many lives the player has, or the HP of bosses, the default control, etc. Use the video as a reference. Again, you are NOT required to copy the video except when asked to, but you may use it as a helpful reference. This will be our "requirement stage" for the project, so that you and I can have a common understanding on what you are working towards, and what my expectations are.

II. The key features **not required** for **Deliverable 1** but you should **start planning ahead** are:

1. The video is a "no shoot" play-through, meaning the player did not fire a single bullet. This is intentional to show the entirety of the stage. In a regular play-through, the player can fire projectiles as well to kill the enemies (hence reduce the number of enemies/bullets and shorten the time needed to survive).

2. The player was hit a few times in the video (02:56, for instance). This happens when the player's hitbox collides with one or more bullet's hitbox.

3. You are NOT required to imitate the details, such as movement patterns/bullet patterns from the game for this deliverable. But you should get yourself familiar with your chosen game engine by trying to copy it. This is a good exercise on working with MonoGame or libgbx.

4. Enemies and Bosses are killed when their HPs (health points) are reduced to 0, causing by damage from the player's projectiles. Bosses may have a separate HP for each stage of attacks. The player on the other hand, functions on a life system as described in the requirements of **Deliverable 2**.

5. Additional tips: in Touhou, projectiles fired by regular enemies remain on the screen when their launchers are killed or exits the screen – unless a Boss Fight starts. On the other hand, the Bosses' projectiles disappeared immediately after each stage ended (either by timeout or losing HP to 0). You are not required to duplicate this behavior for now. But to make your project similar to Touhou, you are advised to implement this in **Deliverable 2**.

The features above required defining hitbox for projectiles, enemies/bosses, and players. For this project, you should duplicate how Touhou hitboxes work, details of which are included below in the requirements for **Deliverable 2.**

III. The features **not required** for **Deliverable 1** are:

1. None of the aesthetic elements is required: music, sound effects, sprites, background, stories/dialogues/characters, etc. For deliverable 1, you may simply use different colored shapes to represent the entities in the game. A big blue circle as the boss, for instance, is good enough. Of course, don't turn in a blank/black screen as your submission!

2. Scores, power-ups (dropped by the enemies/bosses or when the player was killed), bombs, and other

supportive systems.

3. **Max of 5pts extra** are available for the teams who are able to have a functional game regardless of design that is very similar to the actual game. This will be judged based on my game play experience only on the original game and yours. In another word, arts and music/sound do not matter.

4. Project Deliverable 2: see Due Date on Canvas (45pts)

Requirements: The main objective of Deliverable 2 is to refactor, redesign and re-implement what you have done in Deliverable 1, with some new required features. You are not required to start all over and rewrite every single line of code of course, but the amount of work will depend on the design quality of your first deliverable. In addition to imitate the game play in our sample game play video, the new requirements for Deliverable 2 are listed below.

I. The key features **required** are:

1. The **hitboxes** shall be in place and **collision detection** shall be functional to allow regular game play. **(2pts)**
   a) The player should be able to fire projectiles and damage/kill enemies and bosses using the keyboard, and the player should be able to be killed by the enemies' and bosses' projectiles.
   b) For the player's projectiles, your implementation can be as simple or complicated as you want. I suggest keeping it simple, and not interfere with the enemies'/bosses' projectiles, i.e. use different textures to avoid confusion, etc.
   c) For enemy projectiles, hitboxes vary depends on the sizes and shapes of the bullets. You are free to define your own in boxes, circles, ellipses, and so on with varying sizes. Note that a laser's hitbox is stretched out through its entire length (see 04:06). Typically, the hitboxes of the projectiles are smaller or the same as the sprite.
   d) For enemies and bosses, the hitboxes are much larger and mostly cover the entire sprite. Killing of enemies and bosses are based on HP calculation: the player's projectiles caused damage by reducing the HPs until it reaches 0, at which point they exit and/or disappear.
   e) The player's hitbox is much smaller than the sprite. The player is hit once its hitbox collides with any other non-player projectiles' hitboxes.

2. The life systems for the player shall be in place. **(3pts)**
   a) The player has an initial number of lives that shall be displayed properly.
   b) Every time the player is hit by a projectile, it loses one life immediately.
   c) If there are still lives left (shown as stars on the right side panel behind "Player"), the player respawns from the center bottom screen. It remains invincible for a few seconds after respawning. Being hit also cleared the screen of all bullets for a few seconds as well.
   d) If there are no more lives left, the player loses the game and the game should show proper prompt for the player to end or exit the game.
   e) If the player survives till the Final Boss is beaten (or exits), the player wins the game. The game should also show proper prompt for the player to end or exit the game.

3. The details of the game should be taking shape at this point following your own refined plans from **Deliverable 1**, including: **(5pts)**
   a) Entry/exit of waves of different types of enemies, bosses (timeout should be implemented as well if the bosses do not die in a certain amount of time).
   b) Movement of enemies, bosses, bullets should be fleshed out to make it feel like a real game. There should be at least 3 different types of movements for enemies and bullets, for instance.
   c) Quantities of enemies, bullets, and their patterns.
   d) Win/loss condition should be implemented correctly.

Note that I will grade your implementation of these features based on how well you are using the proper design patterns. Simply get them done in an unorganized manner is not the goal here. See below.

II. Other **requirements** are:

These are not game "features", but are requirements for Deliverable 2.

1. You are required to use appropriate design patterns to implement various aspects of the game. You should utilize the design patterns you have learnt from the course at this point. E.g., at the minimum, these patterns should be considered in your design and implementation: Factories patterns for spawning; Command pattern to handle collision detection; State patterns to handle the behaviors, etc.

   You are free to choose other patterns for the features, as long as they demonstrate consideration for the flexibility of design, which will greatly help with Milestone 3. **(10pts)**

2. Basic concepts of an overall architecture of your game should also be in a reasonable, if not finalized or mature shape. This requires packages/subsystems partitions and interface design, sequence diagram driven implementation, etc. The minimum quality requirement would be decent source code structure (i.e., don't put everything in one gigantic Main class) **(10pts)**

3. A design document that includes the following contents: 1) design quality attributes applicable to the game and your solution; 2) UML class diagrams depicting design patterns you used for the game, and explain the reason for choosing said design patterns; 3) UML component diagrams that shows the architecture of your game – for simplicity purposes, it should be either a multi-layered architecture, or a Model-View-Controller architecture. Proper components shall be defined in each of the subsystems in your architecture.

   Reminder: Your implementation should be following your design, if not 100% identical. **(10pts)**

4. A draft of the JSON file (or in other formats) that will be used for Deliverable 3's game engine. This file should contain information about how the game proceeds. See Deliverable 3 for more details. This is required at this point as it will greatly help with your own improvement, and to get feedback from the instructor. **(5pts)**

III. The key features **not required** for **Deliverable 2** but you should **start planning ahead** are:

1. Power-ups (dropped by the enemies/bosses or when the player is hit), bombs, and other supportive systems are trademarks of such games. You should plan ahead to implement at least one of the supportive systems, which will be required in the final deliverable.

2. An easily activated cheating mode. (Why? Discuss.)

3. Tips: Making abstractions from all aspects of the game you have implemented: enemies, bosses, bullets; movement patterns; etc. Ask yourselves this question: can you easily switch the sprites, movements, bullet firing patterns, quantities of Enemy A to Enemy B, or Bullet A to Bullet B? In Deliverable 3, you will be tasked with designing the level with all possible abstractions you are capable of. Again, the challenge of this task will depend heavily on the quality of your design, so make sure you plan carefully with your implementation.

IV. The features **not required** for **Deliverable 2** are:

1. Aesthetics are still not required at this point, as long as the game is in general playable.

2. The specific values of HPs (enemies and bosses) and damages (player projectiles) are up to your team. Same goes for the actual size/shape of hitboxes of enemies/bosses/bullets. They should make sense in a normal game play though: for instance, making the boss's HP 1 and the player's damage 1000 would not be reasonable. (*Unless you played Undertale.)

3. Another **5pts extra** are available for the teams who construct a functional menu within the game, and allows for basic key configurations. For instance, allowing the player to use WASG instead of arrow keys. It's a simple function, but make sure your solution is "elegant" in your design.

5. Project Deliverable 3: see Due Date on Canvas (40pts)

Requirements: The main objective of Deliverable 3 is to build a level interpreter for the game, that is able to reproduce all AI behaviors in your game, specifically, the type/movement pattern/attack pattern of all enemies, bosses, and their projectiles. In addition, your final product should be completed in all designs and features.

I. The key features **required** are:

1. An in-game engine (or even a stand-alone program) that functions as a "level interpreter" of the game. **(20pts)**

   a) The engine shall be able to read in an external file (i.e. in JSON or other formats as you see fit – for examples, see http://json.org/example.html), and then interprets the file to dictate the AI behaviors throughout one level of the game.

   For instance, you might define a script named "stage1.json", in which you wrote:

```
{
  "waves":
  {  "wave":
     {
        "id": "1",
       "time": {"0", "1000"},
       "enemyType": "A",
       "enemyAmount": "5",
       "interval": "200",
       "enemyBulletType":
       {
          "color": "red",
          "type": "B",
          "amount": "20",
          "speed": "10"
       }
     },
     "wave":
     {
        "id": "2",
        "bossType": "Boss1",
        .....
     }
  }
}
```

   You should be able to load the script from the game and start a level. The "AI" should behave

accordingly to your script by deploying, for instance, "waves" of enemies that moves a certain way, looks a certain way and fires certain types of projectiles a certain way.

This is just an example, and the "wave" definition is mere an inaccurate suggestion on how to represent the level. Make sure to define and design how you want to model your level accurately and effectively. Nevertheless, here is the minimum list of info that should be defined and customizable in your script:

i. The type of enemies to spawn; when/where/how/how many do they spawn, etc.
ii. The type of movements of the enemies/bosses.
iii. The type of bullets the enemies/bosses "fire"; when/where/how/how many do the bullets spawn, etc.
iv. The type of movements of the bullets.

The "type of movements" is highlighted as a hint on how you should think about the abstractions at this stage. Simply put, the goal is to make as many AI-related elements in your game customizable as you can. This means there should be as little "hard-coded" information in your source code as possible.

During the final demo, I will ask you to adjust your script on the fly to see if your design is truly flexible and customizable, and grade your project accordingly. The more customizable the JSON is, it means the more flexible the design is, and higher grades you will get on this.

2. Implement at least one of the following supportive systems (not including scoring system). You are free to define your own rules regarding these systems and you don't have to follow the Touhou rules. (It is a good place to start for inspiration though.) **(5pts)**
   a) Power-ups: dropped by enemies/bosses when they are killed, or when the player dies. The player can pick up the drops and increase the damage and/or amounts of its projectiles.
   b) Bomb system: when the player is in danger, the player can "throw" a bomb to get out of trouble. For instance, the bomb can clear all bullets in a certain area (or the whole screen), deals massive damage to all enemies and/or bosses, grants the player invincibility for a short period of time, etc.
   c) Reward system: the player can sometimes pick up life/bomb pieces to increase its life/bomb counts.

3. Complete a design plan for "Secret features" to be revealed later. Details of this requirement will be discussed in class. You would be required to apply proper design patterns to incorporate these features into the game. These would be about customizable behaviors regarding collision detection, and/or user control. **(5pts)**

4. Your final boss is able to perform and mimic these two attacks specified in the video: one at 01:36 – 02:22, and the other one at 03:07 – 03:52, respectively. Analyze what is happening in the video, and try to copy those attack patterns to the best of your ability, using good design strategy. **(10pts)**

II. The features **not required** but can earn you **extra points** (up to 10pts) are:

1. Aesthetic make-overs: full on sprites, backgrounds, animations, sounds/music, etc. **(3pts)**

2. Easily activated cheating mode. **(2pts)**

3. Difficulty level design: for your game, engine and script, find a way to add additional difficulty levels that the player can play. **(2pts)**

4. Actually implement the secret features mentioned in Deliverable 3, I.3, in addition to complete the required design plan. **(5pts)**

5. Other additional features you come up with that you think are worthy of rewards.