Prof. P. Sen **ECE 184 Introduction to Video Game Development** Spring 2025

Final Project Specifications and Deliverables

As we've been discussing, in this class you will work on an **individual final project** where you will each design and develop a **fun, high-quality video game intended for public release**. While you have tremendous flexibility in terms of what kind of video game you can make (e.g., you're free to choose the genre of your game, what it's about, the mechanics, gameplay, story, characters, etc.), there are some constraints to ensure everyone's project has comparable scope and effort.

This document formally outlines the specifications for the final project, many of which have been already discussed in considerable detail in class. It also spells out the **different project milestones and their deliverables,** so please read through the entire document before you start working on your project! Also, before you begin, check out this page for useful tips on how to make a great intro-class game!

I. HIGH-LEVEL SPECIFICATIONS

Here are some high-level specifications your project must adhere to.

1. Independently developed

You will have to **independently** design and develop your game, so this will be an **individual project**. I want everyone in this class to have the experience of working on a game from start to finish and to implement every aspect of the game so that they know how *everything* was done. As we all know, team projects don't guarantee this, and there are many team projects where some students do all the critical work and some do very little. I don't want this to happen in this class.

The key reason for this is that I want everyone who moves onto the advanced class to have gone through the entire game-development process so that they understand how everything works. At that point, they would have a better understanding of what they like (and don't like) about the game-development process, and can drill down further in that class and specialize a bit more. That can't happen unless everyone is familiar with every aspect of game development.

2. 3D game project

Your **game should be a 3D game**, meaning that it should use 3D assets and Unity's 3D engine (i.e., "Universal 3D"). So you can't make a 2D game using 2D textures and sprites. However, although it should not be a 2D game, the bulk of the gameplay can be in 2D. For example, you can make a side-view platformer or an "isometric" top-down game where all the action predominantly happens in 2 dimensions, as long as you're using the 3D pipeline and there are aspects to the game that make it clear it's 3D (for example, you

have levels that are not just a plane but have some height variations to them, cut scenes/effects that leverage the 3D engine to show your 3D assets, etc.)

The reason for this requirement is that working on a 3D game gives you more flexibility for future development: it's easy to work on a future 2D game once you've done a 3D game in this class, but it's much harder to work on a 3D game if you've only worked on a 2D game! So for that reason, we're asking everyone to work on a 3D game for the class project.

3. 30 minutes of fun play

Your game should provide the player with **at least 30 minutes of fun, entertaining play**. Of course, your game can be (much) longer than that, but there should be enough content to keep the player engaged for at least 30 minutes. The reason for the 30-minute requirement is three-fold. First of all, this is the typical amount of time that it takes someone like me to assess a game and tell whether it's fun or not. Second, this is a typical time for a game demo, so think of your project as being a self-contained, free demo for a much longer game you might make in the future (but make sure it's self-contained, meaning that the game project should have a very clear ending!). Finally, it's hard and time-consuming to create games with much longer playtimes than that, so 30 minutes seemed like a good compromise.

4. Playable on fairly low-end PCs and Macs

Your **game should be playable on both PCs and Macs** because students in the class usually have one or the other (make sure that your Unity Hub has installed the modules for "Mac Build Support (Mono)" and "Universal Windows Platform Build Support" to do this). This means that for every milestone (more below), you will need to build and submit two executables: one for Windows and one for Mac (at the end, you can also make a Linux executable but that's optional).

Also, while it's fine to make a game that targets mobile devices (say, to release on the Apple App or Google Play stores) or consoles (which require specialized hardware to test and a different Unity license to build), make sure that it runs well (and is fun) on a PC and Mac and follows all the specs in this document, because that's how everyone is going to test and evaluate your game.

Finally, you need to make sure that your **game runs on even lower-end machines**, because some students have fairly light-weight laptops that can't handle graphics-intensive games. So please don't make a game with super intense graphics that requires a beefy machine to play, because your peers won't be able to play them! A question we invariably get every year is "I have a beefy machine, how do I know if my game will play on a lower-end machine?" Well, the answer is simple: just test it. Surely someone in your cohort group or a friend has a light-weight machine that isn't a super-gamer laptop. So try out your game on theirs! Welcome to the world of releasing actual software products, where you have to test your software on machines other than your own!

5. Rated "T for Teen" or lower

Your game should be rated "T for Teen" or lower (i.e., "E for Everyone", "E10+ for Everyone 10+") by the ESRB ratings standards. So please no intense violence, blood and gore, sexual content, etc. The reason for this is two-fold. First, remember that your student peers are going to be playtesting your game to evaluate it and give you feedback, and many people might be uncomfortable playing games that have a lot of mature content. It's unfair to make them do that, plus they might not enjoy playing your game! So don't do that!

Furthermore, our goal is to release the games to the public, and if you do it under the Red Athena contract, the UCSB name would be attached to the game that is released and UCSB will work to promote the games as well. So for obvious reasons, the university wants to make sure that games released in that manner reflect our values. So please, make sure your game does not have mature stuff.

However, to be clear, this project is considered a **scholarly work**, and as such it can be "edgy" or push the envelope and motivate people to think beyond their comfort zones. So we don't have to automatically self-censor, but it must be done in the right way! Please talk to us if you have an idea for a game that you think might be questionable or raise concerns.

6. Free public release

The plan is to release most (if not all) games produced in the class on public platforms (that's kind of the point of the class!). So whether you decide to release your game with Red Athena or individually by yourself, you will need to create an itch.io page for your game and develop the marketing materials to market your game (more on this in a separate handout).

Furthermore, for simplicity and legal reasons, your game should be entirely free-to-play (F2P). So it will be put on itch.io for free and should not have any in-app purchases, etc. Of course, you are welcome to extend the game later and make a for-pay version, where this project would be the free demo for that game. But this is not something to focus on now and can be worked out after the class is over.

7. No specialized equipment needed to play

Your game should be playable with only a **standard keyboard and mouse**. You should not require gamepads or other specific gaming hardware (VR/AR headsets, guitars, dance pads, etc.) The reason for this requirement is practical: the teaching staff and other students in the class will need to playtest your game to evaluate it, and many of us don't have any of these special devices! So while it's fine that you design your game to be used with, say, a dance pad, it should be playable (and fun) with only a keyboard and a mouse because that's how we're going to evaluate it and grade it.

8. Playable by one person, even when multiplayer

Your game should be playable by one person by themselves without needing other players to playtest, because not everyone has a friend available to playtest your game with (especially since we're going to be doing so much playtesting towards the end of the quarter!). So while you are allowed to do a multiplayer game, all multiplayer games must have a single-player mode which is how most of us will playtest and evaluate your game, so this single-player mode should be fun and entertaining.

9. Clear to understand how to play

It should be easy/clear to understand how to play your game, without having to read instruction manuals and READMEs. After all, no one reads these instructions—they just start playing! This means that if your game requires non-obvious keys, mechanics, etc. to play, then you need to tell the player this information organically through screen prompts or in-game information. In the very minimum, you can add a "Controls" screen in the pause menu, but this is much less preferred (i.e., you'd get a lower grade) than if you do it organically in the game.

Note that while your game should be easy to understand how to play, it doesn't mean your game should be easy to play!

10. Intellectual Property (IP) and novelty

You must **own (or have the right to use) all the intellectual property (IP)** in your game. So you can't make the next Super Mario Bros. game or a Marvel vs. DC battle game, for example, unless you have the legal rights to use these characters in your game (which I'll bet you don't). And without the rights to those characters, you cannot release the game to the public, which is kind of the whole purpose of the class. This also makes it difficult to create games in certain genres, such as music/beat games (e.g., like *Guitar Hero*) because these kinds of games usually require songs people have actually heard to be fun and enjoyable, and it's usually impossible to get the legal rights to use those songs in our games.

Furthermore, your game must be novel. I don't want you to make a remake of another game just with different "skins." There should be some novelty in your gameplay. It's fine if your game is inspired by another game or that it's from a fairly standard genre (e.g., FPS). But then you need to make sure you have some clear twists in your games (e.g., modified mechanics, gameplay, etc.) that make your game somehow different and unique than games that came before. Remember, you want your game to stand out!

11. Use of third-party assets

On a related note, you ARE allowed to use third-party assets as long as **you have the legal rights to use them**. You can get these from Red Athena, buy them yourself from another source (e.g., the Unity Asset Store), or contract an artist yourself to create them (if you do this last one you'll need a legal contract between you and the artist–even if they're a buddy–that turns over their intellectual property to you). Welcome to the world of public product distribution: everything will need to be above-board and well-documented!

And, since we are planning to release all the games made in the class, **you are NOT allowed to use assets you do not have the legal rights to use**. This includes 3D models, images, sounds, music, text, etc.

Furthermore, to be clear, the only third-party assets you can use are either **artistic assets** (3D models, textures, images, animations, sounds, music) or **software that is not specifically game-dev** (networking, AI, scene loaders, etc.) and hence not the focus of the class. After all, Unity itself is basically a third-party software asset that does a lot of stuff (like getting user input, loading 3D models, rendering, etc.) for us so we can focus on actually working on the game logic. So that's appropriate use of a third-party asset.

However, **you are not allowed to use third-party assets for stuff that a game-dev software engineer is supposed to do**. For example, there are many starter toolkits in the Unity Asset Store that allow you to build a specific kinds of games quickly and easily with minimal coding because they do the bulk of the logic for you (they usually brand themselves as "easy" and "customizable", because all you have to do is change the models and get a "new" game). Basically, they are designed so that non-programmers can also make some nice games. There are lots of such kits out there for different genres, such as Tower Defense games (or this one), Action RPGs, FPS, etc.).

I want it to be very clear that **you are not allowed to use these kinds of software toolkits**! After all, using such defeats the whole purpose of this class, which is to teach you how to write stuff like this yourself! We are familiar with many (if not most of these starter kits) and if we find that you have used one for your project **you will likely fail the class**! We often have a student or two who thinks they can get away with this, but they are caught and pay the price. So don't try it.

If you want to know what is allowed and what is not, ask yourself a simple question: is this something that an engineer working in game-development would be expected to be able to produce themselves? If the answer is yes, then you need to do this yourself. If the answer is no, then more than likely you can use a third-party asset. For example:

Would a typical software engineer working in game-dev:

- Compose music for the game? No, so you can use third-party music
- Model 3D assets for the game? No, so you can use third-party assets
- Create animations for fight moves in the game? No, so you can use mo-cap libraries of fight sequences
- Write networking code for the game? No, networking is not a game-dev specific thing so you can use a package for networking so you can focus on the game
- Create a tower-defense logic? Yes, absolutely. So you're not allowed to use a third-party package that basically implements the core of a tower-defense game.

If you have questions or feel like there might be any doubt, please ask me to double check. Note that you will need to document all the third-party assets you use in your game, and in some cases you may need to acknowledge them in a README or credit screen with your released game.

12. Grading

Your project will be graded according to the "Final Project Grading Rubric" which focuses on quality, fun, originality, playability, and effort. So your game should excel in all of these categories in order to get the best possible grade. Please see the rubric document on Canvas for more information.

II. LOW-LEVEL SPECIFICATIONS

In addition to the high-level specifications, your game should have the following features:

1. Run in Full-Screen Mode

After you double click on your application executable, your **game should play in full-screen mode**, not windowed. It should also be able to adapt to displays with different resolutions correctly. So, for example, the UI should not be broken when viewed on different machines!

2. Use the Universal Rendering Pipeline

Your game should use Unity's Universal Render Pipeline (URP (Universal 3D), not the High-Definition Render Pipeline (HDRP) which is more taxing than URP and significantly limits the number of devices your game will run on. Also, HDRP typically demands more complex assets than URP, which is usually beyond what we can do in this class.

3. Splash Screens

Your game should show **Splash Screens** when the game is launched so that it looks professional. You will find a Unity package in this directory to do this, so please follow the instructions provided

to incorporate the Splash Screens into your game. If you're releasing the game through Red Athena you will use the Red Athena Studios, UCSB, and Gaucho Game Lab splash screens, but if you're releasing it on your own you can only use the Gaucho Game Lab splash screen (legally, we're not allowed to use the UCSB marks without a license and/or permission from the university).

Note that the package comes with a flag that allows you to skip the splash screens when you enable it. We recommend that you set this during development so you can skip the splash screens every time you want to test your game. Don't forget to disable that at the end for the final "Gold Master" version.

4. Start Menu

Your game should have a high-quality **Start Menu** (also called "Start Screen" or "Main Menu"), the first thing the player sees after the splash screens that features your game's title logo and have at least the following three buttons: "Start" (to launch the game), "Credits" (to show the credit screens), and "Quit" (to quit the game). You may choose to have other buttons as well (e.g., "Settings", etc.). The Start Menu should **set the tone for the game** and get players in the right frame of mind to play it. Also, try to make it look as good as possible, because first impressions matter!

5. Pause Menu

Your game should have a Pause Menu (or "ESC Menu") that you get to when you hit Escape (ESC) while in the game. At the very least, the pause menu should have the following buttons (you can call them something else, if thematically appropriate for your game): "Return" (to continue playing), "Main Menu" (to return to the main menu), "Quit" (to quit the game altogether) – for the two last options ("Main Menu" or "Quit") make sure you ask the player if they are sure they really want to do that in case they hit it by accident). This is very important, because there are many low-quality indie games out there that do not have the option to quit the game in the middle and you have to kill the process to end the game (that's a bad user experience!).

You may choose to have other buttons as well (e.g., "Inventory", "Settings", etc.), depending on your game. You might call this a "Pause Menu" but it could also be called something else like "Inventory" or whatever.

6. Victory/Defeat Scene

Your game should feature some kind of victory scene/screen when the player wins the game, if appropriate. This is basically something to reward the player's effort, so think about what you'll show here. This can be a cut-scene, some text cards, fireworks, etc. Furthermore, your game can also have a defeat screen, if appropriate.

7. Credit Screens

Your game should have Credit Screens where you will put down your name (you want to get credit for this game!) and acknowledge the help and support you got for the project (you might have several credit screens). Feel free to add everyone who helped you make the game!

The Credit Screens should be shown at the end when the game is won after the victory scene, but should also be accessible directly from the Start Menu (not everyone will be able to finish your game, but everyone should be able to see who made it!). After it is done, it should automatically go back to the Start Menu.

The Unity package for the Splash Screens also contains the end Credit Screens, along with a template for making them. Please use this template to ensure they all look consistent across all games (and if you release your game through Red Athena there will be a Red Athena and UCSB screen as well).

8. Varying levels of difficulty

If your game is too easy/hard, this might not be a great experience for the playtester. So if your game is one that might be too easy/hard, then consider having control over the level difficulty (say in the Start Menu or through a control panel in the Pause Menu) so that players can choose how easy/hard they want their game experience to be.

9. Cheat Mode

That being said, even in the easiest mode most of the games made in this class are reasonably hard to play, and it would be difficult for someone like me (who is a total n00b at most games!) to see all the features of your game in a short amount of time. For example, your game might have an amazing boss battle at the end that you're very proud of, but it could actually be quite difficult to get to the final boss, so 99% of the playtesters wouldn't get there in the limited time that they have to play the game!

For this reason, please create ***cheat modes*** for your game that allow us to experience the full game without having to do everything a normal player is expected to do. In other words, allow the player to hit a secret key (or a combination of keys), that gives them the ability to skip to a selected level, get extra ammo, gain HP, become invincible, reveal enemies, etc. In other words, allow us to cheat to make it easier to play your game so that we can see/experience all the cool things you've designed.

The usual way people do this is to have a special key that brings up a cheat menu with buttons/options that allow you to select what cheats you want to do. Please include a README file in your deliverables that includes how to use the cheat mode.

III. PROJECT MILESTONES

To complete your project, you have to achieve the following milestones, where you start by building the minimum viable product (MVP) – the core game that demonstrates the key functionality and gradually flesh it out through repeated iterations until you have a full, high-quality game. To this end, here are the milestones you must complete. **The actual deadlines for each milestone can be found in the "ECE184 Deadline Calendar" on Canvas.**

1. Proof of Concepts (POC#1 and POC#2)

The purpose of a **Proof of Concept (POC)** is to answer key questions about your game and minimize your risk (the unknowns) about your game as much as possible. Basically, what is the **million-dollar, "show-stopper" question** that if you can't answer your game will just not work? What are the things that you don't currently know whether they'll work or if you can pull them off? What is the biggest potential showstopper that would prevent you from creating a hit game?

Clearly you need to know these things ASAP, because if you can't pull them off, you might need to switch to a different game right away! That's the exact purpose of the two POC stages in this project. Usually, this means testing some core game mechanic, see if your game is "fun", make sure you can build the levels or populate them to make the world interesting, do the things that you'll need to make your game work, etc. Of course, the exact critical questions you need to ask will vary from game to game. Some examples include:

- For a high-speed platformer based only on speed-jumping onto dynamic platforms: Can you make the controls tight enough to ensure a good experience for the player? Will it be too hard/too easy? Will you be able to create levels that are sufficiently interesting and engaging?
- For an FPS set in an alien planet: Can you make enemies with interesting behaviors? Can you create interesting/exciting levels for the game? Where are you going to get all your assets from?
- For a puzzle game about a detective trying to solve a murder inside a house: Can you make the puzzles interesting and just the right level of difficulty (not too easy, not too hard)? Can you populate the house with interesting stuff? How will the player know what they can interact with?
- For a game proposing a new version of chess with new combat rules: What are the rules that will make this a fun game? Will it be balanced?

So think about the critical questions that you need to ask for your game. Then, while working on the POCs, figure out what is the smallest playable demo/game (minimum viable product) that will answer your critical questions. For POC#1 start with the most serious questions (largest risks), and then after you're done iterate again and identify the next-most serious questions for POC#2. Each of the POCs should set out specifically to answer these questions as best as possible. Essentially, for the POCs you need to have at least:

- A basic "toy" program that answers the critical questions about the game
- Core mechanics implemented (rough is fine) so we can start getting a feel for the game
- Instructions on how to play your game can be in an `Instructions.txt` file (please include in deliverables)

And you don't need (although it's fine if you have them):

- Complex, animated characters (basic capsules are usually), unless one of your critical questions is "where are you going to get animated characters from and can you get them to work in your game?"
- Any polish to the game (no sounds, FX, etc.)
- An implemented UI, unless that's part of your critical questions

Deliverables: The deliverables for this project submission are the typical deliverables described in Sec. IV.

However, before you start your first POC#1, you should first record a short video blog (see Sec. IV "Deliverables") introducing yourself, the concept of your game, and maybe a bit about the challenges you face ahead. It would be useful to have an introductory video like this to post first before your other "progress update" video blog posts.

So, to be clear, the **POC#1 submission should include two videos** (one from the start of the week and one from the end of the week). All other submissions will only have one!

Grading Rubric:

- Effort (70%): How much effort was put into the submission, as compared to a typical weekly homework assignment?
- POC (30%): How good was this as a POC? Was it informative? Were you able to learn stuff from the process? Did you answer your major questions?

Note: If you decide after the POCs that your game has a fundamental problem and want to switch to another game, please reach out to us immediately so we can discuss the strategy moving forward.

2. Game Design Document (GDD)

After working on your POCs, you should have a better idea of what your game is actually about (which may have evolved away from your original game idea). So now you need to flesh out your GDD-Lite into a more proper GDD.

Deliverables: The deliverables for the GDD are the same as those of the GDD-Lite (One Pager, Product Backlog, Sprint Backlog), but the only difference is that the Product Backlog is for your ENTIRE project now. Furthermore, the One Pager should have hopefully been refined from your original GDD-Lite.

Grading Rubric:

- Thoroughness (30%)
- Core gameplay and mechanics clearly described (25%)
- Production plan (backlogs) (25%)
- Writing (20%)

3. Prototypes (Proto#1 and Proto#2)

Next, you will build upon your POCs to flesh them out into Prototypes that start resembling a proper game. Your first focus should be to add "depth" to the game (i.e., make it playable all the way through from start to finish using the core mechanic that you explored in the POC). This way, you can start gauging the fun of your game and refining the gameplay/mechanics to make it as fun as possible before you start expanding the breadth (adding more enemies, more quests, more levels) or adding polish/frills (sounds, FX, etc.). Essentially, for the Prototypes you need to have everything for the POCs, plus at least:

- Significantly fleshed out the gameplay and mechanics so the player should have a good sense for what the game is about by now
- Significant increase in complexity in the game from week to week (as appropriate):
    - More user actions
    - More enemies

    - More polish on the overall application (e.g., basic capsules have been replaced with actual assets, some sounds added, some FX, etc.)
- Added basic UI components
- Instructions on how to play your game can be in an `Instructions.txt` file (please include in deliverables)
- Start compiling list of third party assets you are using in your game and keep doing this until the Gold Master release.

However, you don't need (although it's fine if you have them):

- Final assets for everything (some of them can still be temp)
- Polish everywhere (you can still be missing some sound effects, FX, no music, etc.)
- A proper beginning or a win/loss termination condition

Deliverables: The deliverables for this project submission are the typical deliverables described in Sec. IV.

Grading Rubric:

- Effort (70%): How much effort was put into the submission, as compared to a typical weekly homework assignment?
- Prototype (30%): How well have the gameplay and mechanics been fleshed out? How close is this to being a full game?

4. Pre-Alpha Release

The Pre-Alpha release is the first full-game deliverable (the first feature-complete build). This means your project should be a fully playable game at this point from start to end. Basically, think of whatever you would submit for the final project in any other undergraduate class, and that's the level of the pre-Alpha. Essentially, for the Pre-Alpha Release you need to have everything for the Prototype, plus at least:

- Made considerable progress from Prototype Releases and significantly increased complexity, in particular (as appropriate):
    - Gameplay and mechanics have been polished and considerably improved
    - Levels are almost fully fleshed out
    - Enemies are almost fully fleshed out
    - Expanded the game
    - UI more refined and polished
- Have a complete game with a proper start and end, and be fully playable from beginning to end
- Game should provide at least 15 minutes of play
- Game should have the final release title
- A basic Start Menu, perhaps with some temp assets
- A basic Victory/Defeat Scene when player "wins/loses"
- A cheat mode to allow people to play all aspects of your game, as described in the specifications
- The game should now be fun to play!
- Instructions on how to play your game can be in an `Instructions.txt` file (please include in deliverables)

Deliverables: The deliverables for this project submission are the typical deliverables described in Sec. IV.

Grading Rubric: Starting with the Pre-Alpha Release, your project will be graded by the Final Project Rubric posted on Canvas.

5. Alpha Release

The Alpha release is the first build suitable for outside testing. Indeed, this is the first build that will be officially playtested outside your cohort group. Essentially, for the Alpha Release you need to have everything for the Pre-Alpha Release, plus at least:

- Made considerable progress from the Pre-Alpha Release and significantly expanded the game, in particular (as appropriate):
    - Gameplay/mechanics that is smooth and close to final
    - UI should be close to final
    - Levels have been expanded
    - Enemies have been expanded
    - NPC dialog (if any) entirely fleshed out
- Incorporated the feedback on the Pre-Alpha Release from in-class discussions and comments from the teaching team
- Expanded the game to provide 25 minutes of gameplay
- Start Menu should have final assets (remember to make your game title logo)
- Victory/Defeat Scenes have been properly polished
- Splash and Credit Screens properly integrated, although Credit Screens might still have temp elements
- Instructions on how to play your game should now be properly incorporated into the game

Deliverables: The deliverables for this project submission are the typical deliverables described in Sec. IV.

Also, at this point you should **start working on your marketing materials**! See handout on Canvas on what to do for this. Although these are due the Sunday after the Gold Master submission, **do not leave this to the last minute**!

6. Beta Release

The Beta Release should be the close-to-final game. For this deadline, you should basically pretend that this is your final submission, and do everything you can to make it as polished and as complete as possible. Everything should be at full-release quality, and it should play/feel/act like the final game. Essentially, for the Beta Release you need to have everything for the Alpha Release, plus at least:

- Made considerable progress from the Alpha Release and significantly expanded and polished the game, for example (as appropriate):
    - Expand, expand, expand!
    - Levels have been expanded and should be close to final
    - Enemies have been expanded and should be close to final
- Incorporated the feedback on the Alpha Release from peer-testing, in-class discussions, and comments from the teaching team
- Expanded the game to provide 30 minutes of gameplay (basically, at this point it should basically be the full game!)
- Have all sound effects in place
- Have all FX incorporated
- Incorporated music into the game
- All writing is correct and clear
- All game assets are finalized
- Added cut scenes or additional sequences to help tell the story

- Credit Screens are finalized
- Executable is correctly named and has proper icon
- Company Name and Product Name set correctly (in `Edit > Project Settings > Player` on Windows, `Unity > Project Settings > Player` on Mac). For games released through Red Athena Studios, this should be the company name.

However, it is okay if there is still stuff that you want to improve (make the boss fight a bit more interesting, add a few more puzzles, add some more enemies). You can do this for the Gold Master Release version.

Deliverables: The deliverables for this project submission are the typical deliverables described in Sec. IV, plus the following additional deliverable:

- Draft of itch.io page (provide us with "Secret URL" link if not released through Red Athena)

7. Gold Master Release

This is basically the **final, shipping version** of your game. Everything should be in place and ready to go! Essentially, for the Gold Master you need to have at least everything for the Beta Release, plus:

- Made considerable progress from the Beta Release and significantly improved the game by making it bigger and better, for example (as appropriate):
    - Expanded the game further
    - Levels expanded further to make them better and more interesting
    - Enemies expanded further to make them better and more interesting
- Incorporated the feedback on the Beta Release from peer-testing, in-class discussions, and comments from the teaching team (there will inevitably be stuff that you need to add/fix/tweak in the Beta, so never assume the Beta is your final version — it won't be!)
- Game is fully polished
- No bugs!
- Game is ready to ship!
- Work to finish up Marketing Materials (due Sunday after GM deadline)

Deliverables: The deliverables for this project submission are the typical deliverables described in Sec. IV, plus the following additional deliverables:

- Your Unity project directory (all assets, files, etc. — everything required to build your game)
- Separate "cheat" builds (call these "`*_cheat.exe`", where * is the title of your game and which is the name of the normal builds)
- A list of all the third-party assets you used in your game and where you got them from (and if necessary, the appropriate licenses showing you have permission to use these assets)

Note that uploading your Unity project directory to Box takes considerable time, so give yourself enough time before the shipping deadline to do this.

Finally, note that you may still want to tweak some stuff in your game before you actually release it to the public. That's perfectly fine! For games to be released by Red Athena and promoted by UCSB we will work with you to help finalize the game for shipping. Remember that your first official release version should be version 1.0.

IV. DELIVERABLES

The core set of deliverables for all the milestones (except for the GDD) will be the same from now on. For each milestone, please upload to the appropriate Box directory for the specific project milestone (linked on Canvas) the following items:

- Your game builds from the latest sprint in both Windows and Mac format for testing. These builds should implement all what you had set out to do for that sprint! Make sure your game size is minimized by removing unused content from your game when building your executables (expected to be around 300-400MB).
- Videos of your game that show your progress and highlight the specific things you accomplished in the past sprint.
- A `README.txt` file that explains everything, what the videos show, what you got done, etc.
- An `Instruction.txt` file that explains how to play your game, if needed (POCS – Pre-Alpha only)
- The files specified in the Agile document (Sprint Document + PB + SB for the next sprint(s)).
- A short (1 min) video-blog recording where you talk about your progress for that sprint. See "Making Video Blogs" below.

Note that the releases from Alpha onwards have additional deliverables as stipulated in Sec. III. For all project submissions, please zip up everything (no .rar files, please) and name your file:

`<NameOfGame>_<LastName>_<FirstName>_<Proj#>.zip`

Where `<Proj#>` is should be "Proj5" or whatever the number for the project submission is (see Syllabus or Canvas). We will use this convention from now on for all project submissions because most people only know the name of the game, not the name of the student who is making it.

Making Video Blogs

The best way to market your game is to release a series of videos where you talk about your game development process as you work on your game. But this is not something you can simply leave to the end of the process, so I want you to **get started immediately**. Basically, I want you guys to record short (1 minute) videos every week where you're talking about your game. What is working so far? What's not working? What are you most excited about? What are your concerns? The point of this is to give the outside world a glimpse into your game-development journey, as you start from scratch and build what will hopefully become a great game.

The best way to do that is to chronicle your game-dev journey with a set of short videos that can be posted on social media in the weeks prior to launch to help promote your game. If you have signed the Red Athena contract, these will be posted and promoted through the UCSB Gaucho Game Lab and Red Athena social media networks, potentially also by UCSB. And if you didn't, you can still post these on your own to promote your game. Remember, doing this not only promotes your game, it also serves to promote you!

Please check out this Red Athena tutorial on how to record good video blogs. You can also create additional video materials, such as recording game video as you're creating your game, or additional video blogs. Remember, this is your chance to tell the world what you're working on!

Please let us know if you have any questions, and good luck with the final project!