

Modernizing College Courses: A Practical Application of Gamification

A Technical Report submitted to the Department of Computer Science


Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Branden Kim
Spring 2020

On my honor as a University Student, I have neither given nor received
unauthorized aid on this assignment as defined by the Honor Guidelines
for Thesis-Related Assignments

Signature ____Branden Kim____ Date __5/04/2020____
Branden Kim

Approved  Date __05/04/20__
Mark Floryan, Department of Computer Science

Modernizing College Courses: A Practical Application of Gamification

Branden Kim
bk2dh@virginia.edu

Mark Floryan
mrf8t@virginia.edu

May 3, 2020

Abstract

Collegiate courses have been around for decades and with them a certain teaching style to transfer the information from professor to student. However, with the advent of new teaching methods such as Coding Bootcamps, online courses, and private tutoring that argueably can save money and time, can college courses deliver the same or higher value to students? It is crucial that collegiate courses keep up with alternative forms of learning as most of them apply gamification principles already, which might explain why they may have higher value. This report aims analyze a new way college courses are taught to students by applying gamification principles along with a choose-your-own-path style of learning. The goal of the application was to increase student knowledge retention, satisfaction, and passion for diving deeper into the topic of the course. This report will dive deeper into the current problems with college courses today in addition to analyzing a potential solution that was implemented that can categorize steps to learning. In the end, there were successful aspects to the implemented solution that show promise to increasing knowledge retention and passion such as students working on what they enjoy and a dynamic style of learning that had a clear feedback reward loop that can be applied to any courses in the future.

Keywords: Gamification. Courses. Feedback-Loop

1 Introduction

What is the purpose of college? In my experience and talking with my colleagues, many have decided to pursue higher education less for the sake of acquiring more knowledge, but rather because it is the logical next step towards a "successful" life. This is one misled step that many people, myself included, have taken, and I would hope that throughout taking multiple college courses, I would enjoy what I learn and be able to retain the information throughout my life. Currently, there are many alternatives to the traditional college degree, especially in the computer science domain. The number of online resources are limitless, for

example now there are 6 month in-person coding bootcamps, online learning platforms such as Udacity, and free blog and tutorial posts scoured throughout the internet. From my experience talking to those who have taken these alternative forms of learning, I noticed that they can give a better amount of value for time as they are significantly cheaper, have job guarantees, and adjust the material logistics according to the student's needs. To that end, I believed that in order for college courses to keep up, some kind of restructure in the logistics or the material given was necessary.

All the courses that ended up being unsatisfactory lack an overall goal, a positive feedback loop, freedom of choice, and a form of personalized attachment. Throughout, this report will outline the goals, explain the implementation of a potential solution, analyze the data, retrospectively explore the results, and investigate future recommendations. As for the implementation of a potential solution, CS4730: Computer Game Design was revamped in a way where students can choose their sub-topic of interest, dynamically create their own future assignments, and collaboratively work together in order to create a complete 2D game. Mainly, this method focused on allowing students to choose what they wanted to focus on in order to increase passion, create a positive feedback loop, and increase the amount the student learns.

If this method gains enough data to be able to significantly improve passion, knowledge retention, and value, the general template of this implementation could possibly be applied to other courses, alternative forms of learning, and even in industry for new-hire onboarding or professional development. By increasing the amount of productivity and amount of learning, this could reduce costs in industry for the time spent on learning and increase employee skill level.

2 Main Objectives

At a very high level, courses in general should have two goals: make sure the student gains the information that he or she needs and try to inspire the student to dive deeper and proactively learn. From my experiences from the courses that I have taken, I noticed that a lot of them fail to deliver because of one or several of the following reasons:

1. The objective of the course is lost
2. The organization of materials seems sporadic
3. The assignments don't assess student understanding well
4. Students can't focus on their strengths

Some of my experiences with college courses have left me not feeling like I grasped the materials as well as I should because, for example, the assignments were too hand-holding or the connection of a certain topic in the course to the course objectives is unclear. After analyzing my experiences on why I was dissatisfied with the current iteration of courses, I outlined a way to logistically design a course so that it wouldn't fail any of the above reasons.

When trying to learn something new, there has to be several key objectives that have to be met:

1. Have a clear overall goal that all material adheres to
2. Have some form of a positive feedback loop
3. Allow the student to choose assignments / materials based on interest
4. Have a form of personal connection for each concept

Some of these objectives were applied from the concept of gamification which is "The idea to harness the motivational potential of video games by transferring game design elements to non-game environments"¹

By game design elements, the article is referring to those elements in a game.

<i>Element</i>	<i>Synonyms</i>
Rules	Framework of agreed rules, constraints, rule-based
Competition	Competitive play, artificial conflict, competitive activity, contest among adversaries
Goals	Pursuit of a goal, goal-directed, objective
Outcome	Unit of scoring, quantifiable outcome, variable and quantifiable outcome
Decisions	Manage resources
Emotional Attachment	Value assigned to outcome, effort invested for influencing outcome

Figure 1: 6 elements to games Hinske et al. (2007)

As outlined above, there are 6 general elements to games. In our implementation, we expanded upon the some of these elements, which will be covered upon more in the next section. Overall, we mapped the elements within most game designs to course infrastructure in a way that retains the overall goal and information of the course.

3 Implementation

Planning out the new implementation of the course took around 3 months. Previously, the course logistics was similar to that of other courses. In the first half of the semester, students would spend each week working on a particular feature of their game engine such as working on the collision resolution. In the second half, the students would form groups of 2 – 3 and start utilizing one student's engine to create a playable game that could be demoed to the class at the end of the year.

¹Sailer, Hense, Mayr, and Mandl (2020)

This course structure by itself is very good, but it could be improved.

In order to adhere to the objectives and elements outlined in the previous section, the logistics of the course as well as create some additional assignments. As a high overview, the first two weeks had the students learn the very basics of a game engine like the game loop and rendering by utilizing AffineTransforms. However, afterwards the students would pitch game ideas and break up into teams implementing the most popular pitched ideas. Each time had an engine, level, and design sub team that was made up of around 2 to 3 students. Every week, each subteam had to complete some sort of feature that would contribute to the overall goal implementing a video game from scratch utilizing what was learned in class.

3.1 Goal

Similar to that of the previous iteration of the course, we made the overall goal of the students to understand game design principles, fundamentally how the game engine works, and what goes behind the scenes. In the division of the assignments and the material of the course, we tried to make everything contribute to the goal of understanding how video games are designed and programmed. However, to add some competition and incentive to create a superior game, we set up the teams so that they will compete for the best game at the end of the year.

3.2 Feedback Loop

A positive feedback loop is essential to the learning process as it builds momentum in the student on what they are doing correctly and what they shouldn't be doing. Similar to that of progress checkpoints in video games there needs to be a reward system that provides intrinsic feedback to the person. By intrinsic, this means something that relates to personality or emotions as one could argue that grades can be utilized as a feedback loop, but I would argue that it is not valuable enough to the person. Instead, we decided to group the students such

that a group's assignments are utilized in another group's assignments, meaning if one group did not finish their assignment or completed it poorly, that groups teammates would suffer and also feel some pressure from the other groups. Furthermore, we tried to incentivize being ahead of groups by offering the chance to complete extra features that could potentially improve the team's game or could replace other assignments. Due to the lack of time and necessity of additional features, we couldn't add the incentives of replacing assignments, but this feedback loop had a positive effect on some students as one was able to create additional engine graphics features such as orthogonal projections.

3.3 Dynamic Progression

An essential component to video games is the aspect of choice. Without any choice or personal say, a person can be taken out of an experience. To allow for the aspect of choice, we decided to allow students to "choose" their own path within 3 sub-teams:

- Engine
- Level
- Design

Each sub-team would have weekly assignments to complete features that contribute to the course goal. Furthermore, towards the end of the semester, each sub-team was allowed to either follow the scheduled assignments or create their own assignments to complete.

3.4 Personalization

I believe that the way information is retained over long periods of time is to create personalized connections to the information. For example, when I learned

about how the AffineTransform worked in relation to getting global coordinates, I created this visualization of the matrix being passed on to the children components that allowed me to internalize the information. Each course assignment should be personally internalized by a student so that it solidifies their understanding. In our implementation, we made each team to submit a demo showing off their feature each week and have a different team present to the others every week. This way students would have to personalize the course material in the form of their own creative demo and show off the demo to other classmates which adds to the feedback loop.

As mentioned before, we decided that allow students to choose a path that most aligned with their style would be the key to applying the Gamification principles. In our very first iteration, we divided the teams in the following way:

	Teams	Engine	UI	Special Effects	Lvl Prog	Char Prog	Design	Testing
Introduction	Week 1	- Game Loop - Subclass Pattern - Sprite / Animation - Frame Rate Cap	- Game Loop - Subclass Pattern - Sprite / Animation - Frame Rate Cap	- Game Loop - Subclass Pattern - Sprite / Animation - Frame Rate Cap	- Game Loop - Subclass Pattern - Sprite / Animation - Frame Rate Cap	- Game Loop - Subclass Pattern - Sprite / Animation - Frame Rate Cap	- Game Loop - Subclass Pattern - Sprite / Animation - Frame Rate Cap	- Game Loop - Subclass Pattern - Sprite / Animation - Frame Rate Cap
	Week 2	- Affine Transform - User Input (Key / Button) - Time 24 games	- Affine Transform - User Input (Key / Button) - Time 24 games	- Affine Transform - User Input (Key / Button) - Time 24 games	- Affine Transform - User Input (Key / Button) - Time 24 games	- Affine Transform - User Input (Key / Button) - Time 24 games	- Affine Transform - User Input (Key / Button) - Time 24 games	- Affine Transform - User Input (Key / Button) - Time 24 games
Sprint 1	Week 1	- Observer Design Pattern - Controller Input	- V1 Basic menu screen - Play, settings, etc - V1 End of Game screen	- Tweening - V1 Music (Sound class) - Horizontal Background Scrolling	- V1 Overworld Theme, Zones, Connections - Come up with multiple ideas/proposals - Meet with Officials for Feedback	- Come up with multiple actual character and enemy ideas / proposals - Meet regularly with Officials for feedback and update	- Tell story through gameplay through environments - Come up with the overall game storyline and mechanic ideas / proposals - Meet regularly with Officials for feedback and update	- Learn methodologies of testing software - Report on testing games - Test existing games
	Week 2	- V1 Collision Detection - V1 Physics	- Add transitions on input to screens - Show feedback that it works - Implement basic tool pop-up	- Basic Image Processing - Camera Control	- Implement changes based on feedback - Finalize decisions to one agreement	- Implement changes based on feedback - Finalize decisions to one agreement	- Implement changes based on feedback - Finalize decisions to one agreement	- Use testing FW (Catch?), create basic tests - Write test for other teams to easily use the testing FW
Sprint 2	Week 1	- Implement Sprite Sheet Use - Implement ECS	- Create Developer Tool Screen (without functionality)	- Particle System	- Implement 1/4th of the entire game levels	- Implement character physics / fuel areas (PAC)	- Design Overworld with Themes, NPC areas (PAC)	- Unit Test Engine Sprint 1 Week 1 Functional Test Special Effects S1W1 End-To-End Test UE S1W1
	Week 2	- V1 Framework API (rendering / moving things into classes) - V2 Collision Detection (multiple hitboxes, types of collision) - Messaging system (Events V2) - State Manager	- Implement menu that shows asset information and allows for change - Shading	- Shaders Part 1 - Lighting - Shading	- Implement 1/4th of the entire game levels	- Implement state machine for states that the player can have - Come up with mechanics / gimmicks the game revolves around and with enemies	- Create Music themes for each zone and color palette - Come up with mechanics / gimmicks the game revolves around and with enemies	- Unit Test Engine S2W1 Functional Test Special Effects S2W1 End-To-End Test UE S2W1
Sprint 3	Week 1	- V2 Collision Detection (multiple hitboxes, types of collision) - Messaging system (Events V2) - State Manager	- Implement Developer Tool Screen - Functionality	- Shaders Part 2 - Post Processing	- Implement 1/4th of the entire game levels	- Finalize V1 of character development and programming	- Create Music themes for each zone and color palette	- Unit Test Engine S2W1 Functional Test Special Effects S2W1 End-To-End Test UE S2W1
	Week 2	- V2 Physics (Rigid Body Dynamics)	- Implement Debug Menu (terminal commands)	- Post Processing Part 1 - Anti-Aliasing	- Implement 1/4th of the entire game levels	- Implement V1 enemy AI	- Oversee S2W1 level and char prog	- Unit Test Engine S3W1 Functional Test Special Effects S3W1 End-To-End Test UE S3W1
Sprint 4	Week 1	- V1 Networking (Socket Programming and Multiplexer)	- Implement breakpoints	- Post Processing Part 1 - Framebuffer	- Implement 1/4th of the entire game levels	- V2 of character development and programming	- Oversee S2W1 level and char prog	- Unit Test Engine S3W1 Functional Test Special Effects S3W1 End-To-End Test UE S3W1
	Week 2	- V2 Collision Detection (AABB vs OBB)	- V2 Start Screen / HUD / End of Game	- Post Processing Part 2 - Multiple Effect Affs	- Implement 1/4th of the entire game levels	- Touch up and finalize	- Oversee S3W1 level and char prog	- Unit Test Engine S4W1 Functional Test Special Effects S4W1 End-To-End Test UE S4W1
Sprint 5	Week 1	- V3 Physics (ice, low gravity, wind)	- V2 Start Screen / HUD / End of Game	- Post Processing Part 3 - Multiple Effect Affs	- Implement 1/4th of the entire game levels	- V3 of character development and programming	- Oversee S3W1 level and char prog	- Unit Test Engine S4W1 Functional Test Special Effects S4W1 End-To-End Test UE S4W1
	Week 2	- Last minute Engine fixes	- Final bug fixes	- Final bug fixes	- Final bug fixes and touch ups	- Final bug fixes and touch ups	- Full Game Test Runthrough Report	- Unit Test Engine S5W1 End-To-End Test character and enemies
Sprint 6	Week 1	- V2 Networking (cloud-based) PAC - Collision Manager	- Final bug fixes	- Final bug fixes	- Final bug fixes and touch ups	- Final bug fixes and touch ups	- Full Game Test Runthrough Report	- Unit Test Engine S5W1 End-To-End Test character and enemies
	Week 2	- V2 Networking (cloud-based) PAC - Collision Manager	- Final bug fixes	- Final bug fixes	- Final bug fixes and touch ups	- Final bug fixes and touch ups	- Full Game Test Runthrough Report	- Unit Test Engine S5W1 End-To-End Test character and enemies

Figure 2: First Iteration of Schedule

In this first iteration, we divided each team into 8 subteams:

1. Engine
2. UI
3. Special Effects
4. Level Programming
5. Character Programming

6. Design

7. Testing

8. Art

Outlined in each cell is the assignment that they should complete by that week. The way that we connected it is future assignments of on sub team would depend on the implementation of another subteam in an earlier week.

While this was the initial plan, we changed it to a second iteration. Because of the overload of material that needs to be covered in the first iteration as well as the problem of having to deal with teaching how to draw art. Instead, in the second iteration , the teams were divided as:

1. Engine

2. Level

3. Design

Overall the schedule was divided as:

Week	Due Date	Engine Team	SFX/Level Team	Design Team
4	2/7			
5	2/14	Observer Design Pattern (headers)	Scenes and Parsing (headers)	Overall Design
6	2/21		Camera and Sound (headers)	Enemies and Environment
7	2/28	Dev Tool		Area Design I
8	3/6	Spritesheets and Controllers	Parallax and Tweens	
9		SPRING BREAK	SPRING BREAK	SPRING BREAK
10	3/20			
11	3/27	Collision System (headers)	Scene Transitions	Character Programming (headers)
12	4/3		UI Components	Enemy Programming (Enemy Example)
13	4/10			Area Design II
14	4/17	Engine Improvements I	Environment Programming I	Bosses I
15	4/24	Engine Improvements 2	Environment Programming 2	Bosses II
16	4/29	Menus and Polish	SFX and Polish	Character / Enemy Polish

Figure 3: Second Iteration of Schedule

Here we divided each team into 3 subteams where students could choose which sub-team they wanted to join. The rows represents weeks in the semester and assignments further in the semester requires proper completion from corresponding

subteams earlier in the semester. For example, the Level's Parallax and Tweening assignment requires the Engine's Observer Design Pattern for the event functionality. Without proper implementation of the observer design pattern, the assignment cannot be completed. This adds a dependency from the Engine team to the Level team that contributes to the feedback loop and personal connection.

Furthermore, we made the deadlines flexible such that the sub-team can re-submit their work anytime that they wish. This means that if the sub-team didn't completely finish their assignment, they will be able to resubmit a complete version at any time for full points. This is adhering to principle of how games have flexible completion. In a video game, when you fail to complete a level, the game doesn't progress and prevent you from completing future assignments because you failed to complete a previous quest. Similarly, the class shouldn't progress to more advanced topics just because one failed a previous assignment. However, the longer that a sub-team waits to complete an assignment, the longer that the other sub-teams cannot finish their assignment so it creates an incentive to finish the assignments on time.

The overall assignment documents gave a short explanation on the functionality of a feature and gave a high level API overview of what should be completed. This decision was made so that when another sub-team utilized another's implementation, they would be able to know exactly what API's they needed to use to utilize the features of the previous implementation.

Here is an example of one of the documents:

Towards the last few weeks, we made each team submit an Alpha, Beta, and Final version of their game. During these submissions, the teams were allowed to either work on a feature that we have previously outline for them or create their own feature as an assignment. Especially in the final submissions, we thought that students would have created some personal attachment to the work that they have done so far and would like to extend features that are necessary.

PART 1: Create the Classes

You will be writing two classes and two interfaces (or abstract classes). As always, you don't have to follow these guidelines exactly, but I recommend you follow them closely. They are as follows:

NOTE: ALL of these classes should be placed in a new folder ("/engine/events").

- **EventDispatcher:** Equivalent to "observable". Lists the methods necessary for any object that throws events. Contains the following methods:
 - o addEventListener(EventListener listener, String eventType)
 - o removeEventListener(EventListener listener, String eventType)
 - o dispatchEvent(Event event)
 - o hasEventListener(EventListener listener, String eventType).
- **EventListener:** Defines what all listeners must do. Has a single method:
 - o virtual void handleEvent(Event event); //equivalent to notify in many textbooks.
- **Event:** Object encapsulating the information regarding a single event that occurs. Fields and methods include:
 - o String eventType
 - o EventDispatcher source //the object that created this event with the new keyword.
 - o Getters and Setters
 - o **If your event needs more information (like questId, or timestamp, then you would extend this class and add the additional information to that new event type.*

Part 2: A Simple Test Demo

Once you've written these classes. Let's test them out by making a sample game! Let's suppose we have a quest involving a character grabbing an item. Place two sprites on the screen, move one with the keyboard over to the other. Have the coin (or whatever is being picked up), throw some kind of PickedUpEvent and become invisible. Have another class called QuestManager (that keeps track of quests) listen to, hear the event, and print out something like "quest is complete". Yes, this is contrived, but it helps prove that your code works correctly. You will likely use a line of code like this:

```
myCoin.addEventListener(myQuestManager, PickedUpEvent.COIN_PICKED_UP);
```

Where PickedUpEvent.COIN_PICKED_UP is a constant static string that represents that particular event.

Figure 4: Example of an Assignment Doc

New implementation of CS 4730: Computer Game Design	
Gamification Principles	Course Mechanics
Clear Objective	Group up in teams and make a video game with custom engine
Feedback Loop	Peer to peer presentations and praise from teammates
Freedom of Choice	Can choose subteam of choice and desired features in later assignments
Personal Connection	Creation of a demo for each assignment

Included above is a table of the course mechanisms that were implemented and how they relate to the key objectives outlined in the previous section.

While this implementation was in practice, we became something like stakeholders that dictated what the type of game was being created as well as directing the teams in the right direction. In the Retrospective section, I will outline differing aspects from the theoretical schedule and the actual implementation.

4 Data Analysis

In the middle of the semester, a survey was released to all of the students for their overall satisfaction and thoughts on the structure of the course and the materials, in order to measure the satisfaction of students with the new implementation of the course. In the survey, 21 out of 64 students responded. Within this survey, we asked short answer questions on the course structure, rewarding aspects, difficulty, and volume of material. To first analyze students' general satisfaction of the course, I created a box-plot of the overall satisfaction, motivation levels, and structure of the course:

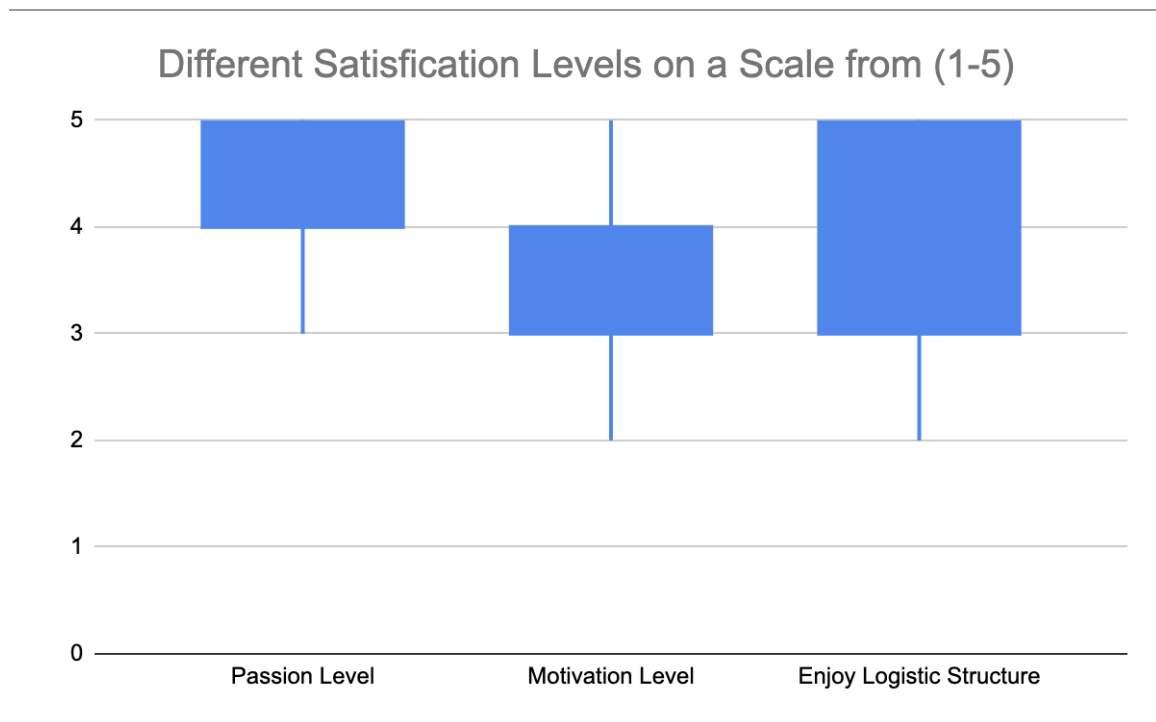


Figure 5: Box and whisker plot of Satisfaction Levels

Here we can see a visual representation of the aggregated data collected from the passion, motivation, and enjoyment of the course structure scaled from 1 – 5. Overall we observe that there is a median of 4 across all of the metrics, but the third quartile ranges from 4 – 5. Analyzing this data, we can see that the majority of the students overall enjoyed the new course structure and found it valuable in terms of their motivation, learning, and passion.

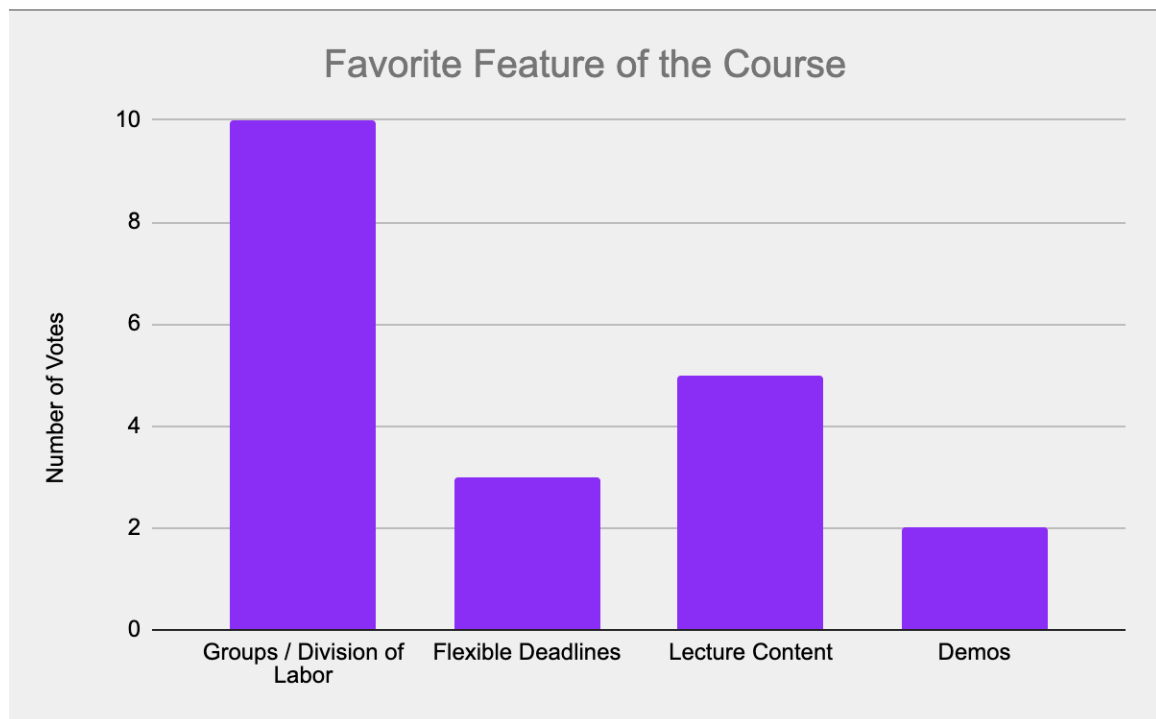


Figure 6: Bar Graph on the Favorite Part of the Course

In this bar graph below, I categorized the responses of the overall course satisfaction in discrete categories to try to analyze what aspect of the course is the most popular among the students. According to one of the students, "The course is flexible and broad - we're covering lots of topics in game design and there is always something to do, learn, think about, and explore. This is reinforced by the homework assignments, which provide a goal rather than an instructive sequence; so you can take numerous approaches on each element. To add, it's awesome that you can focus on a subarea of game design: we still get exposure to each aspect, but can really spend time on topics we're more interested (and motivated to). In that sense, there is a nice balance between course broadness and specificity, instead of just a survey course or a hyper specific portion (like if the course focused a bunch on just on graphics, for instance). The TAs are widely available, albeit I admit I have not used them even though I would like to (I tend not to use office hours practically even if it would be wise). During lecture, we see even more of this; it's discussion based and generalized in a way that encourages students to think about

it on their own.” Based on the data, it seems that a majority of the students enjoyed the sub-teams and having the choice to choose the sub-team of their choice.

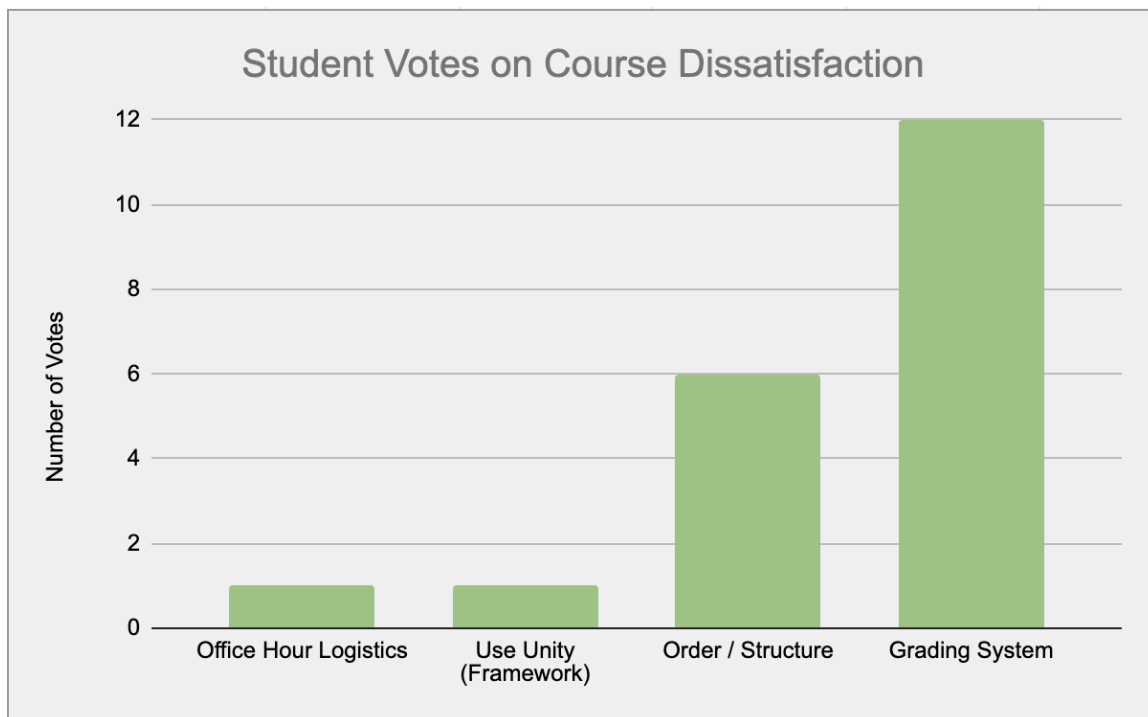


Figure 7: Bar Graph on dissatisfaction of the Course

In this bar graph above, I categorized the responses of the overall course dissatisfaction. Based on the majority of the students, it seems the highest amount of dissatisfaction in the course logistics is due to the grading format.

In this bar graph below, I analyzed the responses on the desire to see more advanced material on the course. One thing that is always hard in balancing course materials is adding advanced material without making the course overwhelming. In this graph, we can see that the majority of students would prefer seeing more advanced material.

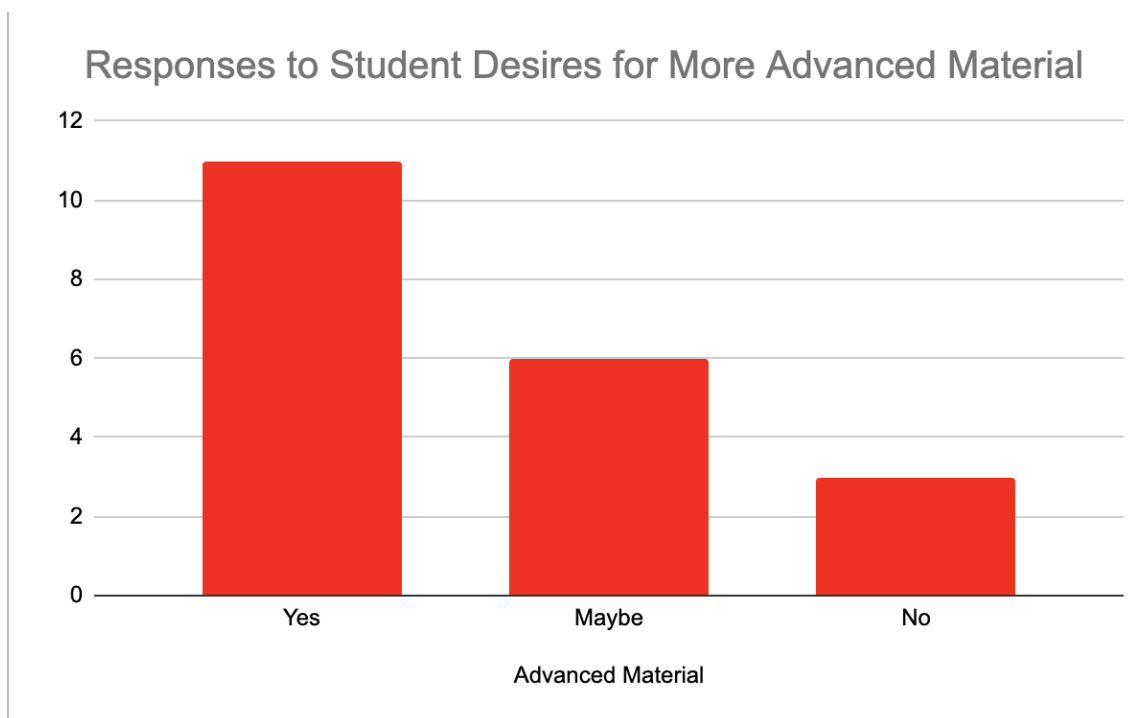


Figure 8: Bar Graph of Desire of Advanced Material

This final graph represents what the students found the most rewarding in the course.

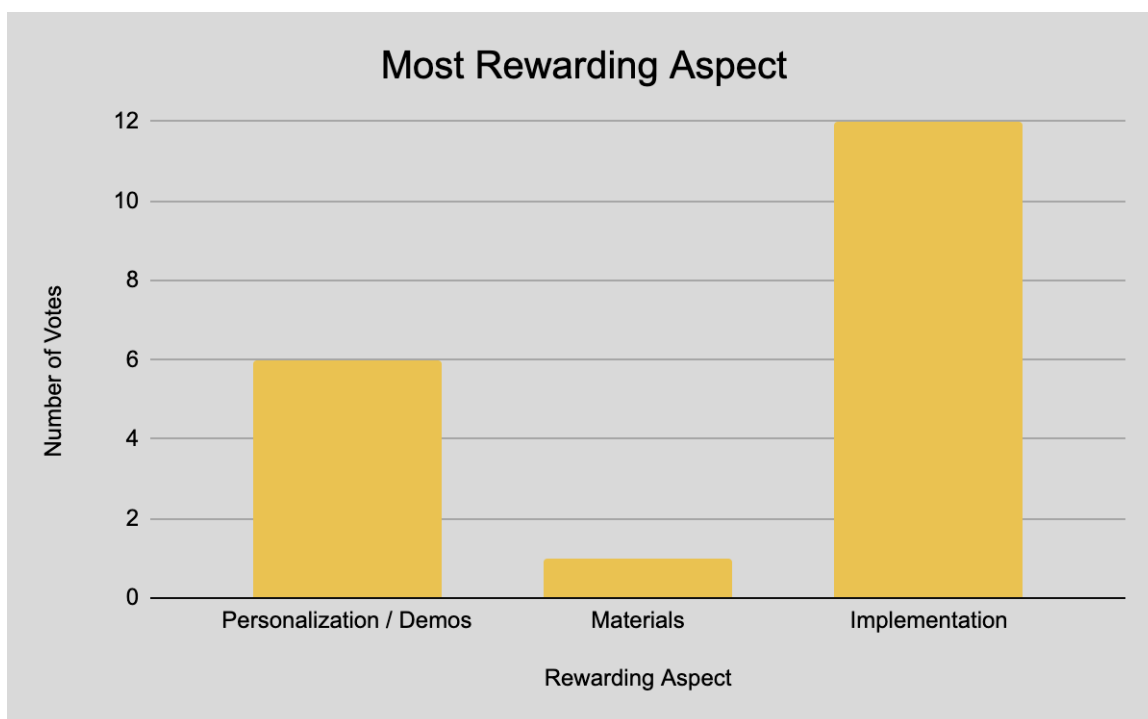


Figure 9: Bar Graph of Most Rewarding Part of the Course

It is interesting that what students found the most rewarding is seeing the fruition of their implementation. Looking at this, the data might suggest that there is a correlation between a positive feedback loop and the difficulty of projects and assignments. This would be similar to that of how more challenging video games feel more rewarding to the player.

5 Retrospective

Overall, this course implementation was a step in the right direction, but had some logistic problems, mainly due to the grading system. One of the general trends found was how the division of teams and aspect of choice vastly increase student's interest and passion. Many students seemed to enjoy the challenging implementation assignments as well as how they were able to work on the sub-teams that interested them the most. Overall, the satisfaction level across all measures seemed to be fairly high, settling around 4 out of 5 based on the student responses. Furthermore, from many of the student responses, it seemed that how in-depth the course material and assignments went into the topic of game design generated positive responses. There might be a correlation with the depth of material and difficulty level with satisfaction and value. Furthermore, the division of the teams into sub-teams that had to collaborate and communicate with each other to finish assignments was a huge positive. Perhaps the demos that sub-teams had to make and the accountability of the quality of their code due to other sub-teams utilizing it had an effect on the overall passion and motivation of the students.

The way that the assignments were split up into 3 separate sub-teams created an relationship that allowed the implementation to meet the main objectives. Each sub-team was in charge of their own assignments and had other sub-teams depending on them. This created a feedback loop dependency where sub-teams felt accountable of the work they hand off. Furthermore, this creates an avenue for a reward system when a sub-team completes an assignment very well with good

code quality and abstraction. When a sub-team abstracts and creates easy to use APIs, it reinforces the reliability of that sub-team and makes future assignments much easier. This type of intrinsic feedback loop is a great solution to keeping students motivated.

As seen in the data analysis, a lot of the students enjoyed the sub-team structure with the ability to choose the corresponding sub-team. This aspect of choice really allows the students to focus and excel on what they want to learn while still adhering to the goals of the course. With the vast amount of features and optimizations that can be made to the game engine as well as the level, it is nearly impossible to teach the foundations and the more advanced topics within one semester. One benefit of the division of the course into sub-teams was that students were able to dive deeper into the particular sub-topic such as having some of the engine teams optimize their collision detection by parsing out objects that are not in the camera scene. Furthermore, because students are working on what they are interested in, the motivation pushes them to learn more.

Finally, we added the requirements for each sub-team to have to create and present their demo to each other each week. Similar to how in video games there are small challenges that require one to show mastery of a new skill or power-up that was obtained, creating these demos requires the students to understand their implementation. Furthermore, implementing a small demo creates a personal attachment to the obtained knowledge. From my experiences and the responses of the students in the course, the personal attachment is what allowed the information to transfer into long-term memory.

This course implementation was definitely not perfect. From the responses of the students and my observations I noticed that there was a key flaw in how the structure of the course was set up. The course had 3 sub-teams with 3 assignments due each week. However, there was only 2 lectures in one week which meant that not all of the information needed to finish an assignment could be logistically covered. Some of the students in the design sub-team expressed frustration that

most of the lecture material covered concepts for the engine and level sub-team but not the design sub-team. Therefore, when the design sub-teams had to design levels and the character, many were left not really knowing how to go about it and as the semester progressed, many of those students wanted to go back and change some of the design decisions they made. This logistical issue is one that we did not think of when planning out the course and it seems will be an issue for any type of course with multiple teams. A few options could be made to remedy this issue.

1. Focus on lecture materials in the overall semester in bursts. For example, in the beginning of the semester very basic, foundational concepts in the engine can be covered for assignments in the engine and some of the level sub-teams. Right after this, the class can shift to covering game design principles, essential components of video games, and what makes video games fun. Rather than covering small increments of ideas every week for each team, instead covering the entirety of one sub-team in a section of the semester may flow better and make the topics seem that they have a sense of direction. This could have an effect on how the information formulates and sticks within the student's mind.
2. Another option is to have solutions for each assignment readily available and easy to integrate with each team's game. One of the main problems of having assignments not done correctly is that it makes the future assignments almost impossible and has students constantly bug-fixing and going back to fix the mistakes. By having hard deadlines and giving solutions to assignments each week, this ensures that students do not fall behind, although it would be more work on the staff since they sometimes may have to come up with custom solutions for each team.
3. There is also the option of splitting the course up into multiple courses. There is either the option of splitting the course up sequentially or in parallel. For

the sequential option, a possible solution is to offer a "beginner", an "advanced", and "integration" version of the course. The beginner version abstracts a lot of the engine and level features out by utilizing an existing framework such as Unity. In this way, concepts that don't pertain to the framework such as game design, art, software and game testing, etc. can be all be covered in this beginner course. Then in the advanced course, the course could cover more of the features that exist in the engine such as the AffineTransform, Lighting/Shader, Event system, collision detection, etc. Finally, there can be an integration course that requires the knowledge learned from the first two courses and requires the students to make a full-fledged video game utilizing the engine and game design concepts learned. Furthermore, in this "integration" course, there are definitely going to be some bug fixing and additional features that may be specific to a game. This can all be finished / covered within this last course. This could work nicely with the semester curriculum as logistically, this works as a concentration similar to that of USC's game development track.

4. For splitting up the course in parallel, each of the sub-teams can be its own courses that sort of run dependent on each other. The engine and game design concepts can be split where the engine course covers implementing the engine features and the game design course covers the design concepts as well as the aspect of building out the game with the engine that is built in the engine course. This necessarily doesn't even have to be split up as separate courses. Instead, the number of lectures in the course can be increased to 4 where the first 2 weeks has the students attending every lecture to get the very basics of game design down such as the game loop and the components of a game. Afterwards, the course will be split up into 2 lectures for the engine and 2 lectures for the game design. Additionally, there can be one extra lab section that is utilized for the game designers and engine people to

collaborate. This way, there wouldn't be a shortage of lecture coverage but also, towards the later weeks in as seen in figure 3, the engine course can focus on optimizations without hindering the implementation of the game and the game designers can just utilize the engine without having to worry about the actual implementation. Unlike the sequential option, this parallel option would be more complicated with the logistics of a college major. The amount of credits and the approval from the department for a course run in this manner seems almost radical.

Another issue that was observed was that the implementations of the assignments by the sub-teams usually came with a lot of bugs or were often incomplete. This adds a lot of complexity to the overall schedule as future assignments are blocked as well as the implementation of the overall game. In the last 3 weeks of figure 3, the teams had to come together and start implementing their game. Theoretically, there should not have been major bugs or incomplete assignments from the earlier weeks, however, due to the flexible deadlines and the lack of customized solutions, many of the teams spent a majority of their time bug-fixing and completing additional features. This was a huge issue since it barely left any time for completing the actual game. To remedy this, there are several options:

1. One option is to reduce the amount of material covered throughout the semester.

This would mean decreasing the amount of features that needed to be implemented so that many of the core engine, level, and design decisions are completed by about halfway through the semester. This way the teams have half of the semester to spend on bug-fixing and building out their game. Even though this gives the students significantly more time, it may not be the best option. The main objectives of the course is to learn about good game design and how the engine works in the background rather than to build out a completed game. By reducing the amount of content covered, the goal feels lost. Furthermore, as in figure 9, a majority of the students found that the im-

plementation of the concepts covered in the lecture was the most rewarding aspect of the course.

2. Another option is to have very robust and correct solutions to the assignments combined with hard deadlines. With this combination even if sub-teams fall behind, the solutions could be provided such that no other sub-team falls behind and by the time the teams have to implement their video game, time isn't spent on bug-fixing and finishing out previous assignments. Furthermore, since the students have hard deadlines that directly affect their grades, students will be more motivated to learn and complete the assignments on time. There are a few negatives to this approach. First, it is going to be difficult to provide custom solutions specific to a team. For example, a team might require a special type of level editor or a specific implementation of physics that, without a custom-made solution, may be detrimental to the team. This would increase burden on the staff as each week they would have to come up with solutions specific for each team. Furthermore, if one of the sub-teams fail their assignment and have the solution handed to them, there is no incentive for them to learn how the solution works and how to relay that information to their team, which would mean that they lose some understanding of the material.
3. Instead of having cutoffs with hard deadlines, another option is to have general solutions with a "redemption" deadline. In this version, general solutions such as an optimized collision resolution, a general dev tool, etc. are provided after an assignment is due. In the case that a sub-team fails to complete an assignment, their "redemption" would be to integrate the general solution and make it specific to their game. For example, if the engine sub-team fails to complete their dev-tool assignment, we would provide a general solution for the dev-tool all set up, but the sub-team would have to integrate it into their solution within the next week in order to get some of the points

back. By giving a partial grade back, this incentivizes the students to finish the "redemption" assignment. This makes the students have to understand how the solution is working as well as figure out how to work with it so that their teammates don't fall behind.

There were also some minor issues with the grading system as well as compilation. One issue that arose in the semester that we did not foresee was cross-platform compilation issues within teams. Some members had a Windows OS, others OSX, others Linux. The version of the C++ compiler created several issues as well as integrating the SDL2 framework with visual studio. Since it was the first iteration of the course, the need for better compile scripts was not foreseen, however, in the future a better compile script is needed such as CMake or Premake.

One of the main issues throughout the course was the grading system. An expressed dissatisfaction from some of the students was how the grading scale was not proportional to the difficulty of the assignment. Huge assignments such as the collision system or tweening that needs to be almost perfectly implemented were weighted the same as very easy assignments such as the observer design pattern for the event system. The lack of scaling may make hard assignments seem not as involved which would cause for problems in future assignments if the feature wasn't implemented diligently. One solution that I believe is necessary in the course is some sort of "skill" tree. The tree tracks the different assignments for each subteam similar to something like:

Similar to the figures above, the skill tree will have 3 different trees corresponding to each sub-tree. In each of the nodes contains information on the assignments, the requirements, and what needs to be completed. Furthermore, each node will have an associated amount of experience points that are awarded on completion. For grades, a certain amount of experience points are necessary for grade cutoffs. Within a sub-team different nodes (or assignments) will have a number of experience points proportional to the difficulty of completing the assignment. For exam-

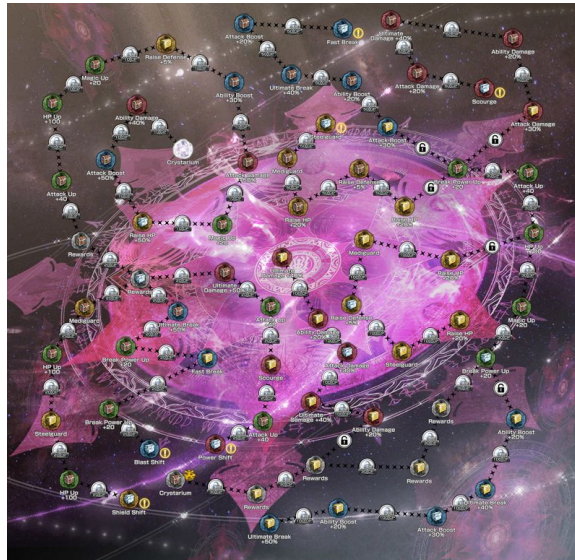


Figure 10: Final Fantasy XIII Leveling System (url: <https://mobius.gamepedia.com/Crystarium>)



Figure 11: Skyrim Leveling System (url: <http://en.uesp.net/wiki/Skyrim:Skills>)

ple, an optimized collision system will have a higher amount of experience points awarded for completion than the basic collision system or the dev-tool will have a higher amount of experience points awarded than the event system. In this skill tree system, the students will be free to choose whatever assignment they want to complete first which helps aid in the aspect of choice as well as decisions can be made based on what is the most necessary for the team. However, certain nodes have a required amount of experience points to be able to see the information and complete. The experience point requirement is to ensure that teams don't only work on optimizing on feature. This requires teams to finish other assignments as well and makes it seem that the assignment completion is more for what is necessary for their team rather than just gaining experience points. In addition, with

the assignment as nodes, there will also be "checks" which represent peer-to-peer demos. As show in figure 6 and figure 9, the peer demos and class presentations was a highly enjoyed. The creation of the presentation adds personalization to the knowledge as well as adds the aspect of competition between different teams that motivates them to work harder and present something unique. These "check" nodes will award experience points and is a nice balance to the course. The skill tree makes it so that it is possible to earn enough experience points to get an A, but just the basic assignments are not enough to receive an A. When faced with that obstacle, a student has the choice to either complete a more advanced assignment or finish the presentation "checks" either of which might take longer depending on the student.

One difficult aspect of the skill tree is its implementation and balance with the proportion of experience points each assignment awards, however one could say this is an aspect of gamification and is a necessary process that constantly requires improvement. The idea of have all the basic assignments open to the student may seem counter intuitive since the lectures flow in a sequential order, but because the first open assignments are very basic, students with some background knowledge or better intuition will still have the ability to complete the assignments in an order different from that of the lecture.

Finally, some students have responded about utilizing a framework like Unity in the course. The subject of whether to utilize a framework or not is one that I see come up quite often in course implementation. I find high value in learning how to implement a version of a framework because it really helps to understand the concepts utilized. However, the only instance that I find utilizing frameworks helpful is when the concepts in the implementation of the framework are beyond the scope of the course. For example, if the course was split up into the beginner and advanced versions, it makes sense for the beginner course to utilize a framework like Unity because the goal of the course is to learn good game design principles as opposed to learning what happens in the engine. The use of frameworks should

only be utilized when the goal of the course does not focus on the technologies utilized in the framework.

As a final outro to the retrospective section, I will offer a more radical idea. The idea is to have students participate in a "battle royale" system. In the beginning everyone has the chance to implement their own solutions to assignments in their respective sub-team. However, as more and more assignments are completed, the ones that do not submit a very good working version or do not complete the assignment are shuffled into teams that end up utilizing frameworks as well as have more basic lecture content offered to them as opposed to more advanced topics. In this system, as time progresses, the best in the class become made up of the best in each sub-team; the best engine implementers and best designers and are the ones that continually push and learn more advanced topics. The ones that are super passionate about the topic have the opportunity to work with others who share the same passion while those who are not can still utilize other technologies and what they consider satisfactory in terms of lecture content to meet the course objectives. This is more of a thought experiment rather than an actual proposition as the logistics might make it very difficult to put into practice.

6 Results and Conclusions

In conclusion, this semester's course implementation of CS 4730: Computer Game Design was one in the right direction. While there were many flaws in the way the course was structured, the course was one of the first that I have seen to divide the content into separate paths and allowed the students the freedom to choose. There are many things that need to be balanced such as the lecture content with the amount of sub-teams, but the course's emphasis on the student's choice with the structure and the peer-to-peer demos made it unique. The main issue was the sheer amount of features and content in the course that couldn't fit into one semester, and I would like to offer some recommendations going forward on

how I would change the course. First, I would definitely want to have the skill tree that I outlined in the previous section to be put into implementation. In terms of the main objectives that I wanted, I think the skill tree encompasses all of it for any course. Furthermore, I would like to have the different teams run in parallel. If the teams are divided in to sub-teams of engine and design, students can either join the engine or the design lectures after the initial foundation lectures and every week there is a laboratory section that is used for peer-to-peer presentations (for the "checks" nodes in the skill tree") as well as "stand-up" meetings for collaboration. I am a huge proponent of the idea that information sticks once it is seen repeatedly. This parallel version of the course can be taken multiple times such that the first time around I could be on the engine side, but the next time I can take the design side of the course. Because the engine and design are sort of opposites, a different perspective to the completing the course objective could be found. Furthermore, the compile scripts would be able to compile cross-platform and basic source control management would be covered. After analyzing how the course went and the feedback from the students, I learned so much and came up with solutions that I don't think I could have seen before. The application of gamification is in the right step into formalizing steps to learning, but I learned how gamification principles are applied have a huge impact on the results.

References

- Hinske, S., Lampe, M., Magerkurth, C., & Röcker, C. (2007, 01). Classifying pervasive games: On pervasive computing and mixed reality. *Concepts and Technologies for Pervasive Games-A Reader for Pervasive Gaming Research*, 1.
- Sailer, M., Hense, J. U., Mayr, S. K., & Mandl, H. . (2020, April). December 23). *How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction*, 23. Retrieved from <https://www.sciencedirect.com/science/article/pii/S074756321630855X#sec2>