Course Title: SDE 205 Game Programming II

Course Leader: David Maruszewski

Expected Learning Outcomes for Course

- Understand principles of video game production
- Create event driven programming environments and algorithms
- Develop game events through the use of C++
- Level and Mod design analysis
- Understand the differences in programming and developing of different genre games
- Adeptly simulate in 2 dimensions and 3 dimensions
- Analyze, select and apply tools appropriate for a specific solution
- Logically formulate scripts and/or programs to solve problems
- Understand and articulate interactivity in the gaming industry, including the connectivity between computer art and programming
- Apply programming and artistic theory in practical applications
- Demonstrate problem solving skills through verbal and written media
- Apply rudimentary Physics and Trigonometry principles

Assessment

(How do students demonstrate achievement of these outcomes?)

Students are required to complete their second project which was created to test skills gained throughout the course. The project is then graded with a "grade sheet" which looks at skills and outcomes. A full assessment rubric may be created in the future.

A supplemental (3rd of the semester) exam is issued to help confirm the findings of the project grade.

Validation

(What methods are used to validate your assessment?)

Currently, all grades sheets are held for two semesters and composite data is used to show trends. Certain chosen questions on exams should help verify or contradict findings.

Results

(What does the data show?)

1. Students can manage coding in project form
2. Error trapping, and creating flawless algorithms is challenging to the student
3. Debugging for some students is very easy, but others seem to be difficult. The split is about 50/50.
4. Being able to explain their programming is something students do not do well. They don't give results. They give commentary.
5. Much material is in SDE 104/205. Students may feel overloaded or we may not get through enough helpful material.
6. Students struggle with the software (UDK).

Follow-up

(How have you used the data to improve student learning?)

1. Projects are to be kept and improved every semester
2. Students will be put in teams in order to help them with debugging and error trapping. This has initially worked well. Test problems were also designed to aid this. Initial assessment of this is good.
3. I pair up stronger students with weaker students to help them out. This can be beneficial to both if done correctly. However, it drains class time. Some material has been put off until homework. The overall content of class seems rushed
4. Initially this has improved. Students tend to be trained to be opinionated in their pasts. A little of un-training is needed. However, getting them to focus on what they are trying to achieve in their code works well.
5. We dropped the C++ component of this course, and taught more to the UnrealScript and generic script reading. The artificial intelligence component was kept. This worked much better, but students don't like the artificial intelligence. Most indicators point to the fact that they find it too hard (scores and words.) I'll keep this under consideration and balance it with the needs of the course.
6. This comes from inexperience and the fact that UDK is a complex beast of a code. Even experts have trials with it. We are thinking of moving to Unity in order to keep academic rigor but relieve unneeded hardships. New text and lessons would have to accompany this. C# would replace UnrealScript.

Budget Justification

(What resources are necessary to improve student learning?)

1. A game engine is used in this class and helps to get the students interested in programming issues.
2. Thawspace would be beneficial for downloading of Open Source material.
3. This was run as a Hybrid, which allowed students to do work from home. Webinar software would work well in this course. I think also that this course should move to a completely online course as one section.