Project II - Personal Research

Goals

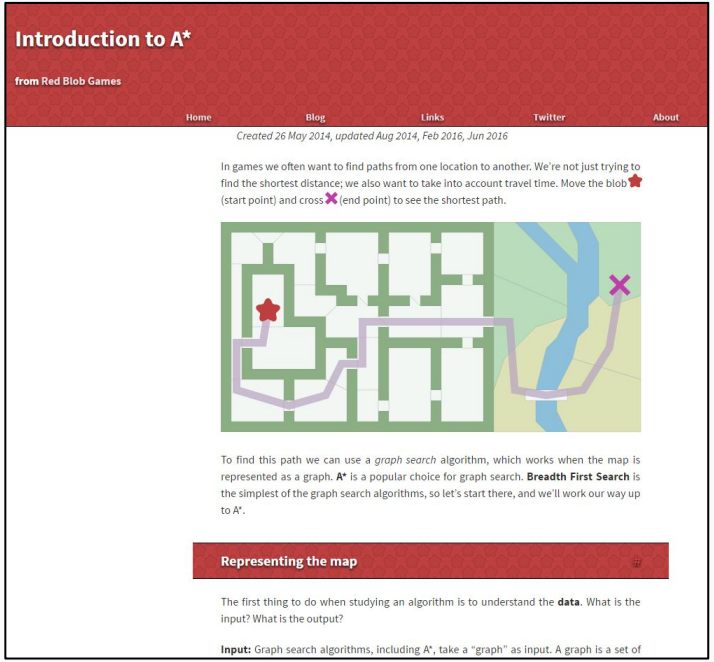
- Learn how to find, solve and assimilate any problem by yourself
- Practice communication performing teacher duties for a day
- Create simple yet generic code that can be used anywhere else
- Have a new element to add to your CV
- Become expert in a small area and attend other classmates’ requests
- Help the games that are being developed with new code modules

Website

- You need to create a website under github.com with:
  - Intro to the problem
  - Different approaches by different games *(use animated gifs when possible)*
  - Description in detail for the selected approach
  - Links to more documentation
  - TODOs and Solution inside the repository as VS projects
  - Optional Homework for practicing
  - Explanation of any other improvements on the system

Website

- The website should be a tutorial that explains the problem and solution with much visuals as you need.
- Focus on simplicity, your audience is not only the class but all aspiring game developers.
- Examples here.



Website

The header of the website should contain the following signature:

“I am <link to your linkedIn>(NAME LASTNAME), student of the <https://www.citm.upc.edu/ing/estudis/graus-videojocs/>(Bachelor’s Degree in Video Games by UPC at CITM). This content is generated for the second year’s subject Project 2, under supervision of lecturer <https://www.linkedin.com/in/mgarrigo/>(Marc Garrigó).”

Pure Theory presentations

- Initial presentations can involve only theoretical concepts.
- Material provided should be **useful** for all teams in their development.
- All student should learn **something new**, avoid explaining obvious concepts that students used to play games already know.
- **Never** copy content without a citation

Code

1. Solution that shows all the TODOs finished. **Only code with STL!**
2. Handout with 3-8 small TODOs from easy to difficult
    a. All TODOs should be easy to find in the code base with comments to support them

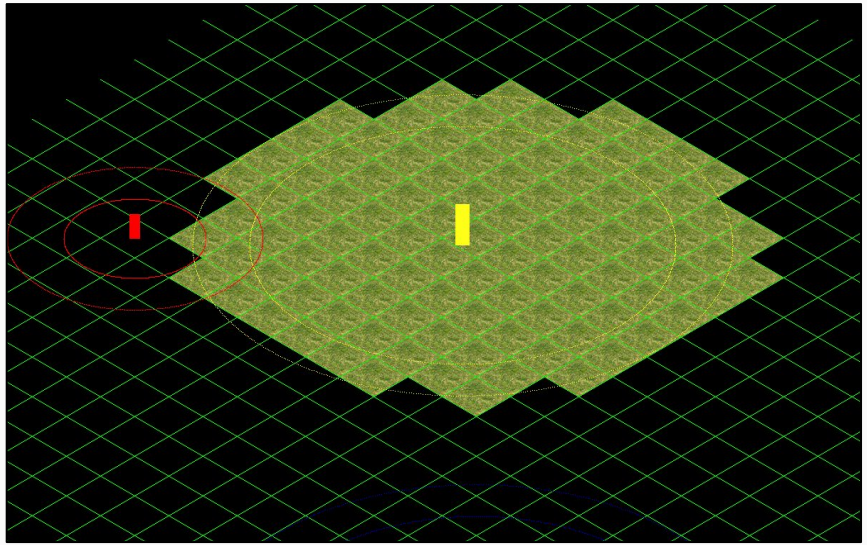
Base code should be as small as possible, striping any other not-used code if possible (same for assets). Code should express the core solution in a **simple** and **understandable** manner. Include as many debug visualizations as possible.

Repository

- Everything should be in a repository:
  - /docs - folder for all website material
  - /exercises - code with the TODOs in comments
  - /full_code - solution (keep the TODO comments)
- It should include a build published in the Release to download as zip
  - Only the executable with everything solved and enough debug visualization to understand the code and how it works (explained in the README)
- Measure your module performance in **ms** and discuss potential optimizations, with links to point out how to do those techniques

Generic Solutions with generic code

GOOD



BAD

Timing

- Teacher will provide a full calendar with all research areas
- Research might be presented to the teacher the previous class before presentation for approval / improvements
- Practice the presentation time and approximate TODOs exercises to last 30/45 minutes
- The material as well as the presentation skills are evaluated in the grade
- Link must be uploaded before starting the class to the proper folder

Research areas

- Teacher can help and guide you *after* you start searching.
- Most common mistakes:
  - Concepts are too obvious.
  - TODOs are too complex / long to finish.
  - Initial explanation of the problem and solution is done too quickly without clarifying well each concept.

RESEARCH AREAS

GAME DESIGN

DOCUMENTATION

CODING

BUILDING

Area: Game Pillars

- Games production always start with a set of 3-8 core pillars
- Those act as filters to discard new ideas during development
- Which game pillars were defined in professional games ?
- What would be a bad game pillar ? How is this useful for our teams ?

Area: Quest Design

- How are quests designed in professional teams ?
- Example of Good/Bad Quest design
- Quest as a learning / tutorial experience
- Dangers of grinding

Area: Boss Design

- How are bosses designed in professional teams ?
- What they mean for the player ?
- How they affect the overall level design ?
- Bosses as a test for players skill
- Boss themes

Area: AI RPG Design

- What different types of AI in RPGs
- What skill do each test
- Scalability

Area: RTS Balancing

- Base for paper/scissor/rock (zero-sum games)
- Expectation of strategic gamers
- How much strategy vs tactics/skill
- Approach to balancing maths

Area: RPG Progression

- Approach from math perspective
- How to hide the infinite loop to the player
- How to avoid repetition
- Balancing issues

Area: Tutorial Design

- Examples of good/bad tutorials
- Learning curve analysis
- First 3 minutes of game
- Tutorial formats used

Area: Production Plan

- Production plan is a document that needs to be delivered
- What it contains ?
- How is it useful ?
- How and when it should be updated ?

Area: Tech Design Document (TDD)

- It defines all technology related items that should be considered.
- Code Style, Platforms supported, code organization overview (UML).
- Branch workflow, code reviews, performance budgets.
- Functionality and limitations expected to communicate to the team.

Area: QA Workflow

- QA uses workflow diagrams to define its pipeline
- Those change from development phase to phase
- Different types of testing
- What about Quality Test ? Stabilization phases ?

Area: Audio Bible

- Audio Bible is a document that should be delivered.
- It mirrors the Art Bible but for audio.
- How it affects the creation of audio ?
- How it defines the production of music ?

Area: Art Bible

- Art Bibles are created by Art directors before entering production.
- Contains guides for both visuals and production limits (texture size, etc.)
- How is this useful for our teams ? Examples of professional art bibles ?
- Items commonly defined in art bibles: palettes ? character references ?

Area: Available Licenses

- Dive into the legal aspect of software and game development
- What is copyright ? trademark ?
- Licensing IPs
- Free software licenses teams could use and their differences

Area: Easing & splines for cameras and UI

- Easing curve library that supports few tween: http://easings.net/
- Spline library that supports common splines
- Camera Traveling / Panning with soft accel: moving A to B via spline
- UI Animations, buttons appearing from the sides

Area: Random map generation

- Dungeon / outdoors map procedural generation
- How to make sure areas are reachable
- Design parameters to meet design intent for each map

Area: Assets ZIP management with PhysFS

- Deploy your game with all assets compressed in one .ZIP/WAD
- PhysFS as a filesystem abstraction layer
- Layers of mounted folders
- Opportunities for encryption

Area: Quest Manager

- How to organize internally player objectives
- Prerequisites: other quests finished ? some stat (level) ?
- Progress track: event system that captures, e. g. inventory changes
- Data driven quest creation with XML

Area: Boost / Power-Up Manager

- Data Driven (XML) system to define properties
- Not only for RPGs, all games will need this
- What if you receive a + 10 HP and then a x2 HP ? And when removed ?
- Can they stack ? Can they run logic “buff that makes you immune to …”

Area: Event System

- System to broadcast events, useful for everything, e. g. UI / collisions
- Specially interesting if you can subscribe to specific events
- And that event sending is delayed until a certain point to avoid chaining
- There is a design pattern for this: Event queue

Area: Dialog System

- Intro to different dialog systems in games, from simple to complex
- Ways to define the data (XML)
- How to remember previous choices ? How to substitute player name ?
- Solid code that can quickly choose between options

Area: Minimaps

- Theory of how minimaps are used in video games (input and output)
- Code should be independent of any entity system
- It should be **interactive** (click on it to navigate) (Ignore Fog of War)
- You should be generating a texture that can be drawn anywhere (UI?)
- Handle circle minimaps , icons and warnings (you are under attack)

Area: Fog of War

- Introduction to all types of Fog of War and why games use them
- Code independent of the game and/or specific entity systems
- First pass with rough edges, *then* smoothing out the edges
- No need for 2D visibility but good to comment it out (check here)

Area: Cutscene Manager

- Wide intro into the pros/cons of different cutscene systems
- Explanation on the current existing cutscene editors
- Data driven code with generic approach
- TODOs to both use and code the system

Area: Particle System

- Intro to particle systems, focus on 2D (check particle2dx.com)
- Explanation on the common particle properties
- At least implement a good looking fire and smoke (not animated)
- Performance warning: Avoid new/delete on each particle

Area: Task Queue

- Be able to chain commands onto units
- Dive into different uses for that in video games
- Useful tool for redefining keys
- Normally achieved using the Command Pattern

Area: Video Player

- Using ffmpeg or ogg vorbis / theora codec
- Decompress a streamed buffer of data
- Every frame dump all pixels into a texture
- Also feed the audio to the SDL audio system
- Explain the codec ecosystem and options

Area: Spatial Audio & Music Manager

- If every entity can potentially generate audio
- Discard the ones that are too far
- If close to the camera, use SDL_mixer to play the audio spatially
- Music Manager: Chain a playlist with fading. Trigger exploration -> combat music and back

Area: Group movement

- Moving troops in groups via complex terrain
- How do we avoid them lining up ?
- How to they handle doors ? and columns ?
- How to they finish the path (avoid all to the same spot)

Area: Split Screen

- Creation of N cameras
- Loop for render different cameras
- Viewport management
- Performance issues

Area: Sprite ordering and Camera Culling

- We are doing it manually so far, but it must be automatic!
- Also, we do not want to Blit elements outside the camera
- In the end we should only order the entities in the camera, there the synergy between those two systems

Area: Spatial ordering optimizations

- Checks against all (brute force) units is too slow:
  - Camera culling
  - Area of Effect attacks
- For static entities we must use a **Quadtree**
- For dynamic entities we must use a **AABB Tree**
- Create both classes and compare the amount of checks and ms spent

Area: Camera Transitions

- Fade to Black is too simple!
- What if we could have a general manager to switch scenes ?
- Supporting many ways for transitioning
- Transitioning also means moving the camera around with curves

Area: Input combos

- The system should have enough memory to allow pressing the buttons over a certain period of frames to consider it a combo
- How could we support a big potential combination of keys defined by the designer ? Collision of combos ? How to make a combo easier or more challenging ?

Area: A* Optimizations (3)

- Our basic A* is going to be very slow to move several units
- Explore some of the many optimizations that exist for A*
- The game should always be responsible
    1. Hierarchical Pathfinding
    2. JPS / JPS+ Pathfinding Algorithm
    3. Time slicing and pooling planners

Area: Crash reports

- Crash dumps and minidumps as help to detect late game crashes
- Overview of *signal.h* & *SetUnhandledExceptionFilter*
- How could we send final user crash reports back to us ?

Area: Static code analysis

- Systems for static code analysis to catch more errors in code
- Example for online: coverity
- Example for offline: cppcheck

Area: Branching Policies

- Explain the concept, the benefits and possible drawbacks
- Have all teams setup gitflow structure for their repositories
- Explain very clearly the use for each case
- How to structure QA around gitflow ?

Area: Automated builds CI/CD

- How to config automatic builds with GitHub Actions
- Guide to write scripts, manual copying README files, etc.
- Automatically zip and upload back to github
- Automatic builds and notifications

Area: Installer

- How to create an installer for your game
- Final touches to be fully integrated with windows:
  - Icons for executable, task bar app (sizes and formats)
  - Sign the executable with company, windows certification, etc.
- Test the game in win8/win10 with virtual machines
  - Dependencies (libraries and dll needed)