Final 2 Feedback

• Include relevant INSTRUCTIONS and READMES!
• It is very hard to grade without an indication of where to look
• This can lead to undeserved incompletes
Final 3 and 4

• You should setup a time to meet with your mentor soon, before Friday
• Make Final 3 and 4 rubrics for gameplay
• Write your game
• $$\text{Profit}$$
Deadline Approaching

- Course policy: you must turn in a working version of all projects
- Deadline for incomplete projects is December 20
- Same day as Final V
- Make sure to email the TA staff when you re-turn in!
AI in Games

Ninth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment

- **AI in Game Design**: AI as a source of novel game mechanics and genres
- **AI-Based Production and Authoring Tools**: Behavior-building, design frameworks, telemetry-supported game design, content authoring support, scripting, sketch-based authoring, automated playtesting
- **AI Techniques for Games**: Planning, reinforcement learning, search, neural networks, Bayesian models, evolutionary algorithms, case-based reasoning, constraint programming, utility-based approaches, animation, camera control, tactical/strategic decision making, terrain analysis, opponent modeling, dynamic difficulty adjustment, spatial decompositions, path planning
- **AI Storytelling**: Interactive drama, story generation, character development
- **Autonomous Characters, NPCs, and Virtual Humans**: Personality, emotion, believability, natural language processing, cognitive modeling, crowd simulation, social robotics
- **Procedural Content Generation**: Level generation, progression design, behavior adaptation
- **Commercial AI Implementations**: Case studies, implementation analysis, comparative evaluations
- **AI in Novel Entertainment Applications**: Entertainment robotics, virtual/mixed reality, mobile device games, geo-location based games, games for human-computation
- **Computational Creativity and Generative Art**: Painting, poetry, story, humor, music
- **AI in Games for Impact**: Training, education, intelligent tutoring, games for health, gamification
Computer-controlled decisions

- Rule-based systems
  - kind of programming
  - Specialized languages
- Goal-based decisions
  - single agent
  - heuristics
- Competitive decisions
  - zero sum
  - pruning
- Cooperative decisions
  - general sum
  - game theoretic
Computer-controlled decisions

- Rule-based systems
  - kind of programming
  - Specialized languages
- **Goal-based decisions**
  - single agent
  - heuristics
- **Competitive decisions**
  - zero sum
  - pruning
- Cooperative decisions
  - general sum
  - game theoretic
Planning Ahead

GAME TREES
Game Tree

- Explicit representation of the decisions the agent can make from the current state along with the possible outcomes.
- State change
- Opponent moves
- Moves of “nature”
Game Tree

- Computation relatively easy
- Expand to bottom
- Leaves have values
- “max” up our decisions
- “min” up opponent’s
- “expected value” up nature
- Take best move at root
Tree Complications

- Tree gets big fast
- Exponential in depth
- Tree can be deep
- Won’t reach leaves
Information Sets

- Secret information
- Poker
- Stratego
- Annotate game tree with “information sets”:
  - Game nodes where player has the same history and therefore must make the same decision.
  - Can’t propagate values up from leaves anymore.
  - Solvable via linear programming.
Planning Ahead

SEARCH ALGORITHMS
Single-agent decisions

- Find the cheapest sequence.
- Can build a full game tree.
- End up visiting all possible nodes.
A*  

• With additional information, can prune and avoid expanding some nodes.

• Admissible heuristic:
  • Optimistic estimate of cost to goal

• Always expand node with smallest cost to reach plus estimate to complete.

• Optimal paths using information optimally!
Admissible Heuristics

- Straightline distance
- Relax constraints
- Background knowledge
- MST vs traveling salesman
Pruning Game Trees

- Although we want to exact value at the top of the game tree, we might only need accuracy within a range elsewhere.
- Return:
  - Beta if true value is beta or above (more than enough),
  - Alpha if true value is alpha or below (not enough),
  - exact value otherwise.
Alpha-Beta

- **Max node:**
  - First child: ask for alpha,beta
  - If bigger than beta, return beta.
  - If bigger than alpha, increase alpha.

- **Min node:**
  - First child: ask for alpha,beta
  - If smaller than alpha, return alpha.
  - If smaller than beta, decrease beta.

- Root: negative to positive infinity.
- Not optimal in terms of effort!
Evaluation Function

- Even with pruning, can’t reach leaf.
- Use evaluation function to guess value.

- $P = 100$
- $N = 320$
- $B = 330$
- $R = 500$
- $Q = 900$
- $K = 20000$

- Human expertise.
- Machine Learning can help.
Monte Carlo Tree Search

- A new approach to search problems you might find useful!
- Applications and strengths and weaknesses compared to traditional search methods.
MTCS vs. alpha-beta

- Consider sequences at full depth, but do not consider all possible actions. Heuristic procedure to select promising lines of search.
- alpha-beta: Wide but shallow, heuristic to fill in value below.
- MCTS: Deep but narrow, heuristic to select actions along the way.
Bandits

- $k$ arms. Pulling arm $i$ has payoff $r_i$, on average. Best arm has payoff $r^* = \max_i r_i$.
- Over a series of $T$ trials, goal is typically to minimize regret: $T r^* - \text{sum } r_i$.
- (Best $T$-step payoff minus actual $T$-step payoff.)
Exploration-Exploitation

- If arm is deterministic, very easy! Try each arm, pull the best forever after.
- If arm is noisy, need to balance exploration (get a better estimate of an arm) and exploitation (choose best arm given current knowledge).
- Originally intended to model medical trials.
- Recent resurgence of interest for online ads.
How decide?

- Arm 1: highest bottom
- Arm 2: widest bar
- Arm 3: highest top
- Arm 4: highest middle
Optimism in the face of uncertainty

- Known that optimal regret grows like $\log T$. Can’t do better. Can achieve it.
- UCB1: Choose the arm with the highest upper confidence bound.
- Example of what I call the Pangloss Assumption. Assume you are in the best of all possible worlds. Either you are right or the world sets you straight.
- Contrast with pessimism. Realistic estimate of what is observed. Great possibility of regret.
• Upper confidence bounds

• For bandit that has gotten $n$ pulls, $n_i$ on arm $i$, total reward from arm $i$ is $r_i$. (C is a constant that can be chosen to make the math work.)

• $\arg\max_i r_i/n + 2C \sqrt{\ln n / n_i}$

• Form follows from Hoeffding bound.
Bandits in Trees

• Think of the top node of the search tree as a bandit problem.
• Guided action choices after the first “pull”. Get payoff at the bottom after the roll out.
Upper Confidence Interval in Trees (UCT)

- More successful choice gets more and more attention/pulls over time.
- Smoothness assumption: Relative average payoffs of the subtrees matches their relative max payoffs.
AI

QUESTIONS?
UI principles

• Does it tell me what I need to know?
League of Legends
League of Legends
UI principles

• Does it tell me what I need to know?
• Is it clear how everything works?
Civilization V
Tooltips
Don’t Starve

Pick up Beefalo Horn
Tooltips
FTL
On the other side of the channel there’s another torch, but it’s too far for you to reach with a Fireball. Combine Earth and Fire to create a Fireball. This can then be cast in the same manner as the Earth projectile. Try to hit the torch with a Fireball.
UI principles

• Does it tell me what I need to know?
• Is it clear how everything works?
• Is it easy to use?
Skyrim

AMULET OF STENDARR

Armor: 0  Weight: 1  Value: 196

Block 10% more damage with your shield.
Diegetic UI
Meta UI
Meta UI
Morals

- Be creative
- Make sure the appropriate information is conveyed
- Get lots of feedback
- Don’t be afraid to iterate
- Take CS130!
QUESTIONS?
LECTURE 11
Embedded Scripting
Content Management

- Designer can make content
- Iterate faster and without recompiling
- Potentially reload content at runtime
Representing Behavior

• How to encode AI or map-specific gameplay?
  – Could use Entity I/O
  – Could use behavior trees, FSM, other structures

• Or, we could use code to express complex behavior
Two languages: worth it?

- One language would be simpler
  - No marshaling needed
  - No API needed

- Could just use Python

- Could just use Java
  - Can dynamically load Java source

- Core engine needs to be fast
  - Collision detection
  - Physics
  - Rendering

- Scripts need to be sandboxed
  - Security
  - Convenience
Complementary Features

C++
- Statically typed
  - Maintainable
- Compiled
  - Long build times
- Direct control of memory
  - Good for CPU-intensive code

Scripting language
- Dynamically typed
  - Flexible
- Interpreted
  - Slower at runtime
- Protection from machine details
  - No memory corruption
Lots of choices

- **General-purpose**
  - Python (Jython)
  - Javascript (Rhino)
  - Ruby (JRuby)

- **Designed for embedding**
  - Lua

- **Designed for engines**
  - UnrealScript
  - Game Maker Language

- **Misc**
  - Pawn, TinyScheme, AngelScript, Squirrel, GameMonkey, etc
Jython

- Create an interpreter
- Convert Java objects into Python objects (marshaling)
  - Can also be automatic using reflection
- Run Python scripts

```java
PythonInterpreter py = new PythonInterpreter();
py.set("x", new PyInteger(42));
py.exec("y = x / 6");
PyObject y = py.get("y");
py.execfile("myscript.py");
```
Lua

- Small learning curve
- Almost everything is a (hash)table
- First-class functions
- Light and efficient
- Relatively simple C integration
- Most popular game development scripting language

```lua
players = { "Peter", "Paul", "Mary" }

scores = { Mary = 10, Paul = 7, Peter = 8 }

table.sort(players, function (i, j)
    return scores[i] > scores[j]
end)
```
UnrealScript

- Used in the Unreal Engine (proprietary)
- Static types
- Networking concepts
  - Replication
  - Simulation
- Editor integration
- Function calls can last several frames
  - Each Entity gets its own logical thread (not OS thread)
  - Much easier than callbacks

```cpp
class FlashLight extends SpotLightMovable;
var FlashLight MyFlashLight;

var repnotify bool bIsOn;
simulated function Toggle()
{
  PlaySound(FlashLightSound);
  // ...
}

var() Color CaseColor;

reliable server function Pulse()
{
  Toggle();
  sleep(0.5);
  Toggle();
}
```
Your Own Language

- Take cs173
  - Implement Pyret
  - Learn about language design tradeoffs
  - Mostly ignores performance
- Take a compilers course
  - Focus more on performance
  - Implement optimizations
  - It runs in the fall
- Jump in blind!
Organization

• It’s inconvenient to keep track of a ton of script files
  – Consider integrating script editing into the engine
• Shouldn’t have to hard code which script file to use
  – This you can easily do with 1971 editor
  – …and should be able to do with any level editor
Language Interaction

• API design: how do the scripting language and high level language communicate?

• Some considerations:
  – Security: script should be sandboxed if users can add scripts
  – Utility: script should be useful
  – Abstraction: script shouldn’t have to do legwork

• Designing APIs is hard!
  – Could use some neat reflection tricks to wrap all Java methods
  – Could use a library to simplify the entire process
QUESTIONS?

Embedded Scripting
LECTURE 11

Tips for Final III
Gameplay is Important

- Implement most of your game logic this week
- Playtest early and often
- Use the feedback you’re getting
Final II Playtesting

Your games are playtestable now!