345 Ludic Computing

Lecture 12

Adaptive Games

Simon Colton & Alison Pease
Computational Creativity Group
Department of Computing
Imperial College London

ccg.doc.ic.ac.uk

Overview

- Machine learning for adaptive games
  - Action prediction
  - Reinforcement learning
- Dynamic difficulty adjustment
- Case studies: adaptive pacing and content

Adaptive Games

The Idea

- Any game that adapts to player in some way
    - May involve the use of machine learning
- Almost any aspect of the game can be adapted
    - e.g. NPC behaviour, dramatic pace, content, narrative
- Adaptation may be online (during play) or off-line (between play sessions, across different games)
- To provide appropriate challenge, or to enhance some other aspect of the player's experience

Adaptive Games

The Reality

- Learning in games is unpredictable. There is a risk that the adapted game will be less playable
- Widespread industry scepticism

> "Training takes place as part of the game development cycle, never after the game ships."
> Neil Kirby, GDC AI Roundtable 2001

- Successful adaptation is usually quite simple and carefully integrated into game design
- Recent titles that use 'experience management': Left for Dead 2, Halo 3, Uncharted 2

Action Prediction

- Where will the player go next? What weapon or attack will they use?
- Games can adapt to player's actions by predicting the next action and responding accordingly e.g. to increase challenge or to create a more interesting experience

Batman: Arkham Asylum
http://www.youtube.com/watch?v=TqHYhBF5fi0

Action Prediction

Observed Frequencies

- Actions 1,2,3,...,n
- A very simple method of predicting the next action is to record the observed frequency of each action and pick the most frequent
- Either based on the actions of all the players overall or the current player's actions so far
- This doesn't take into account the sequence of events leading up to the current prediction
  - Suitable for single choices, e.g. early or late attack?
  - Not suitable for complex sequences of actions

Action Prediction

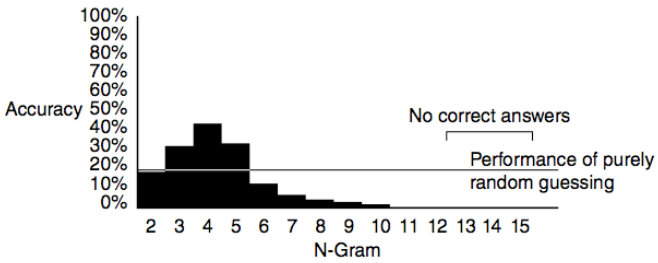
N-Grams

- An N-Gram is a subsequence of length N, for some N > 0
- Record frequencies of action N-Grams, i.e. sequences of length N
  - Observing raw frequencies = recording 1-Grams
- Predict next action based on window W, the sequence of N−1 previous actions. Find the most frequent N-Gram G = Wq for some action q. Then predict q
- $\text{sample}(W) = \sum_{\text{actions } a} \text{observations}(Wa)$
- $P(q \mid W) = \text{observations}(Wq) / \text{sample}(W)$
  - Can avoid making prediction at all if too unlikely

Action Prediction

N-Grams

- Increasing N (and hence window size) typically increases, then degrades performances
  - Large enough window to model action dependancy
  - But too large and overwhelmed by randomness of player's actions
- An optimal N can be established by trial-and-error (alternatively, by statistical tests on action sequences)



5-choice game

Action Prediction

Hierarchical N-Grams

- Larger N-Grams require more training, to ensure all possible window sequences have been sampled several times
- When learning online (i.e. from live player actions) it may take a long time to observe enough N-Grams for our optimal N
- Hierarchical N-Grams maintain N-Gram data for all $N \leq L$, where L is the ideal (optimal) N
- Every action is recorded as a 1-Gram, every pair as a 2-Gram, every triple as a 3-Gram, etc.

Action Prediction

Hierarchical N-Grams

- Initially we will try to predict with a window W of length $m = (L - 1)$, where L is the optimal N-Gram size for this context
- For a window W of length m
    1. Check we have enough (m+1)-Gram data on W
        - sample(W) $\geq$ k for some **sample threshold** k
    2. If so, predict using the (m+1)-Gram data
    3. If not, repeat with window of length $(m - 1) \geq 0$
- N-Gram techniques can be too powerful in some game contexts and may have to be deliberately weakened

Action Prediction

Hierarchical 3-Gram Example

P = Punch, K = High Kick, S = Sweep Kick

Player's input: PKPSSKPPKSPKPKPSPPSSPKKPKPSPPK

3 x 1-Grams
Sample 30

P	14
K	9
S	7

9 x 2-Grams
Sample 29

PP	3
PK	7
PS	4
KP	6
KK	1
KS	1
SP	4
SK	1
SS	2

27 x 3-Grams
Sample 28

PPP	0	KPP	1	SPP	2
PPK	2	KPK	2	SPK	2
PPS	1	KPS	3	SPS	0
PKP	4	KKP	1	SKP	1
PKK	1	KKK	0	SKK	0
PKS	1	KKS	0	SKS	0
PSP	2	KSP	1	SSP	1
PSK	0	KSK	0	SSK	1
PSS	2	KSS	0	SSS	0

Action Prediction

Hierarchical 3-Gram Example

- Window 2, threshold 6
- N-Grams: Window → Prediction (Sample)
- 3-Grams: PK → P (6)
- 3-Grams: KP → S (6)
  - But note that 2-Grams: KP → K (14)
- 3-Grams: PS → P or S (4), below sample threshold
  - So instead use 2-Gram: PS → P (7)
- 1-Grams never used unless we increase threshold

PPP	0	KPP	1	SPP	2
PPK	2	KPK	2	SPK	2
PPS	1	KPS	3	SPS	0
PKP	4	KKP	1	SKP	1
PKK	1	KKK	0	SKK	0
PKS	1	KKS	0	SKS	0
PSP	2	KSP	1	SSP	1
PSK	0	KSK	0	SSK	1
PSS	2	KSS	0	SSS	0

PP	3
PK	7
PS	4
KP	6
KK	1
KS	1
SP	4
SK	1
SS	2

Reinforcement Learning

Overview

- Can adapt a game's actions using a system of rewards and punishments
    - e.g. to learn NPC tactics adapted to a specific player
- General framework covering many algorithms
    1. An **exploration strategy** (or policy) for selecting actions in the game
    2. A **reward function** that gives feedback on each action
    3. A **learning rule** that adapts policy based on feedback

Reinforcement Learning

Q-Learning

- Q-Learning is a popular learning rule used in numerous applications, including board games and NPCs.
- Relies on an **abstract** state machine representation of the game
    - States S, actions A and transitions $T \subseteq S \times A \times S$
    - Non-deterministic: an action in a state may have several possible outcomes
    - Assume a reward function $r: S \times A \rightarrow [-1, 1]$. r may be unstable, i.e. it can return various values for same input
- Every game action a in a state s results in a (possibly zero) reward r and next state s'. This gives us an **experience tuple** <s, a, r, s'>

Reinforcement Learning

Q-Learning

- Stores **quality value** Q(s,a) for each transition (s,a,?) $\in$ T
- Learning is parameterised by **learning rate** $\alpha \in [0,1]$ (alpha) and **discount rate** $\Upsilon \in [0,1]$ (gamma)
- Given an experience <s, a, r, s'> we update the Q-value

$$Q(s, a) = (1 - \alpha)\underline{Q(s, a)} + \alpha(\underline{r} + \underline{\gamma \max(Q(s', a'))})$$

Old value | Reward | Discounted best quality of next state

α blends the old and new (reward + discounted future) values

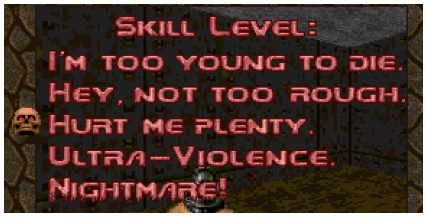
Reinforcement Learning

Using Q-Learning

- Q-Learning is a *learning rule*: we still need a reward function and an exploration strategy
- *Rewards* should be designed carefully: if a small reward is given before a desired state (and large reward) is reached, then Q-learning may learn to continually collect the small reward
- A basic *exploration strategy* is to pick the action with the highest Q-value (with a fairly high probability) or pick a random action (with a fairly low probability)
    - In some contexts the player's actions can be used as an exploration strategy for learning NPC behaviour

Dynamic Difficulty Adjustment

- Adapt the level of challenge to the player's abilities to give a consistently fun experience
- Typically rate player's ability (e.g. mean health, kills, deaths) then adjust game parameters (e.g. damage dealt and taken)
- Can be implemented by a variety of techniques, from a simple formula to a machine learning of suitably challenging behaviours
- Used in some commercial games at least as far back as *Xevious* (1982), but not always documented or publicised



Dynamic Difficulty Adjustment

Examples

- Games vary in what is changed about the game and how much this alters the challenge, e.g.
  - *Max Payne* (2001) adjusted aim assistance and enemy health based on combat performance
  - *Half Life 2* (2004) adjusts contents of supply boxes based on health
- Machine learning can be used to adapt the game, e.g. reinforcement learning can reward NPC behaviour that gives a suitable level of challenge

Dynamic Difficulty Adjustment

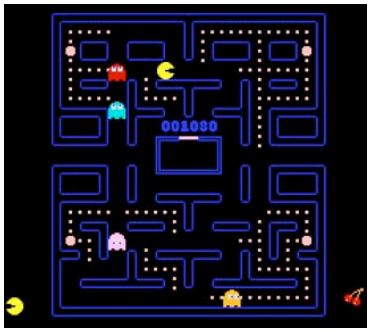
Design Pitfalls

- DDA can ruin gameplay if done badly
- The **rubberband effect** in racing games: do well and your opponents become super fast, do badly and they become ridiculously slow
- Adaptation should not be too immediate, strong or obvious: it can remove any sense of challenge or progress.
- The player should still be able to win or lose, and challenge should vary, with DDA controlling the maximum level
- Players should not be able to easily exploit the DDA, e.g. perform badly just before a end-of-level boss

Dynamic Difficulty Adjustment

Evolving Entertaining Opponents

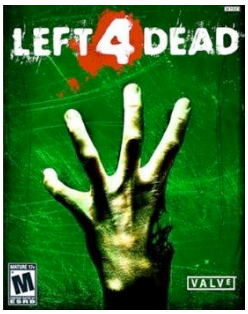
- Yannakakis (2007) evolved artificial neural networks to control *Pac-Man* Ghosts that were more entertaining for a given player
- Ghost ANN fitness based on three entertainment metrics
  - Level of challenge (mean time until death)
  - Behaviour diversity (variation in time until death)
  - Spatial diversity (entropy of location visits)
- Good behaviours evolved offline, then adapted to player online



Adaptive Pacing

Left for Dead (2008)

- Co-operative first-person shooter zombie survival game, with procedural generation of enemies and resources
- The 'AI Director' adapts the pace to the players' activity, creating peaks and troughs of intense combat



1. Estimate each player's **emotional intensity** (a number) and find the maximum intensity
2. If maximum intensity above a threshold, generate no major threats for a while
3. Otherwise, continue to generate threats

Adaptive Pacing

Left for Dead (2008)

"From a dramatic standpoint, ideally we want to see the survivor team just about make it to the next checkpoint, limping in being chased by the horde, so they can celebrate when the door closes and they yell "We made it!"

These breaks... are critical for a couple of reasons. The primary reason is battle fatigue; constant fighting and gunfire is tiring. Secondly, the teams needs an end to these skirmishes along the way to give them a chance to regroup, heal and reassess. Lastly, these big exciting battles are only exciting if there are also periods of quiet, creepy, tension and anticipation to contrast them against. This dramatic pacing is really critical to creating a fun experience."

Adaptive Pacing

Left for Dead (2008)

- Various events increase a player's intensity score
  - Injury by enemy (proportional to damage)
  - Enemy dies nearby (inversely proportional to distance)
  - Pushed off of a ledge by an enemy
  - Incapacitation
- Intensity decays over time unless the player is being attacked

Adaptive Pacing

Left for Dead (2008)

1. **Build Up**: generate full threat population until intensity reaches peak threshold
2. **Sustain Peak**: continue full threat for a few seconds
3. **Peak Fade**: generate minimal threat until intensity decays below threshold. Allows current combat to play out
4. **Sustain Relax**: continue minimal threat for 30-45 seconds after intensity falls below threshold, or until players have moved far enough away from the location of the peak event

3 and 4 provide a "relax" period between activity peaks

Adaptive Pacing

Left for Dead (2008)

2) Mob removed during relax phase

3) Peak intensity not reached

1) Hit peak intensity: sustain then relax

4) Mob added as intensity low

Adaptive Content

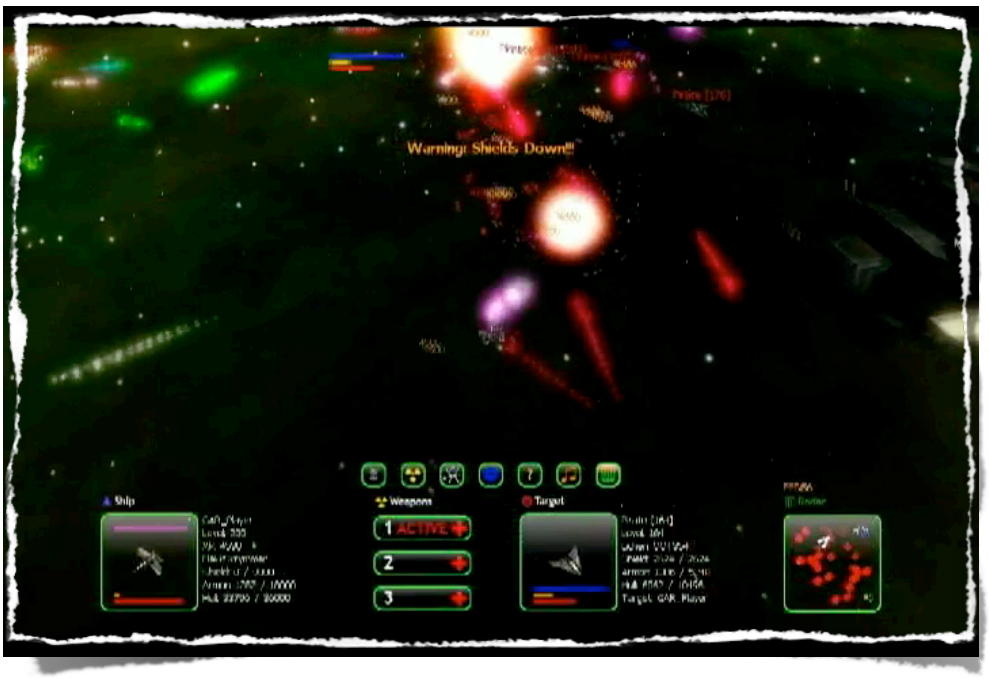
Galactic Arms Race (2009)

- Models weapon projectiles as particle systems
  - Good for fuzzy objects
  - Trajectories + visuals controlled by mathematical rules
- They adapt game content to a player's preferences by evolving new content from the existing content (i.e. weapon) they currently favour
- Particle systems generated via compositional pattern producing networks (CPPNs)
  - These are a type of artificial neural network (ANN)
  - Evolved using neuro-evolutionary techniques (cgNEAT)

http://gar.eecs.ucf.edu/index.php?content=weapons

Adaptive Content

Galactic Arms Race (2009)



http://www.youtube.com/watch?v=MOKNoZSAGGQ

Summary

- Adaptive games have huge potential, but limited exploitation because of unpredictability
- Successful examples are simple and carefully integrated into overall design, e.g. dynamic difficulty adjustment
- Application for machine learning, e.g. action prediction or reinforcement learning
- Case studies: adaptive pacing and adaptive content

Principles of Game AI

(Mat Buckland)

- Often many good solutions for a single game
- Each solution will need to be tailored to the game, and you won't get it right first time
- Don't pick the latest/coolest/most complex AI technique unless the game *really* needs it
- A fun agent is much better than a clever agent
- The sophistication of an agent should be proportional to its life span

Further Reading

(non-examinable)

- Millington & Funge (2005) *AI for Games* (either edition)
- Mitchell (1997) *Machine Learning*, ch.13 *Reinforcement Learning*
  - https://www.doc.ic.ac.uk/~maja/mlcourse395/Mitchell_ML_book.pdf
- *Galactic Arms Race* interview
  - http://aigamedev.com/open/interviews/galactic-arms-race/
- G.N. Yannakakis and J. Hallam: "Towards Optimizing Entertainment in Computer Games", Applied Artificial Intelligence, 21: 933-971, 2007