DM810

Computer Game Programming II: AI

Lecture 14

Learning

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Resume

1. Movement
2. Pathfinding
3. Decision making
4. Tactical and strategic AI
5. Board game AI

Outline

1. Machine Learning in Games
   - Parameter optimization
   - Action prediction
2. Decision Learning
   - Naive Bayes
   - Decision Tree Learning
   - Reinforcement Learning
   - Artificial Neural Networks
3. Evolutionary Computation

Outline

Machine Learning in Games

How can machine learning techniques be used in Games?

- adapt to each player, learning their tricks and techniques and providing a consistent challenge.
- produce more believable characters: characters that can learn about their environment and use it to the best effect
- reduce the effort needed to create game-specific AI

Requires understanding of the peculiarities of each technique.

Classification of Learning

- Offline learning (most used)
  - either between levels of the game (short) or more often at the development studio
  - Use data sets to learn classifiers and approximate functions.
  - optimizes (tunes) values of free parameters in strategies to minimize/maximize fitness
- Online learning
  - While game is running, constantly adjust classifier/approximator/parameters
  - problems with predictability and testing

- Intra-Behavior Learning
  - don't change the whole quality of the behavior, but simply tweak it a little
  - Examples: learning to target correctly when projectiles are modeled by accurate physics, learning the best patrol routes around a level, learning where cover points are in a room, and learning how to chase an evading character successfully
- Inter-Behavior Learning
  - qualitatively different mode of action
  - hopeless to learn everything, other specialized techniques treated are preferred
  - making decisions between fixed sets of (possibly parameterized) behaviors

Pitfalls

- It is impossible to avoid the AI learning the "wrong" thing.
- constrain the kinds of things that can be learned in a game
- overfitting
- balance of effort: work to get the desired result by a learning algorithm vs developing specialized algorithms

Learning Techniques

- parameter optimization
- action prediction
- naive Bayes classifiers
- decision tree learning
- reinforcement learning
- artificial neural networks

Parameter tuning

- example of parameters:
  - magic numbers that are used in steering calculations
  - cost functions for pathfinding
  - weights for blending tactical concerns
  - probabilities in decision making
  - ...
- for each value of the parameter there is some energy value (aka, fitness value, score) representing how good the value of the parameter is for the game
  - compute by playing the game many times at normal speed, without rendering the screen
  - heuristics
- hill climbing: starts random, find direction of improvement, eg steepest gradient
- metaheuristics: random restart, simulated annealing, etc.

Action prediction

- guess what players will do next
- Psychology: humans are bad at behaving randomly
- work out what we'll do or think next based on a relatively small amount of experience of what we've done in the past
- Example: left or right

Raw probability

- count previous choices
- gives a lot of feedback to the player, who can use it to make decisions more random

String matching

- choice is repeated several times
- `LRRLRLLLRRLRLRR`, search the last few choices (window size) and see what usually comes next

	..R	..L
LL	$\frac{1}{2}$	$\frac{1}{2}$
LR	$\frac{3}{5}$	$\frac{2}{5}$
RL	$\frac{3}{4}$	$\frac{1}{4}$
RR	$\frac{0}{2}$	$\frac{2}{2}$

N-grams

- N-gram: sequence of written symbols of length N
- 3-Gram would be a predictor with a window size of two
- keep a record of probabilities for each sequences
- look at the last N-1 inputs to predict the Nth input
  `LHHLHLLLHHLHLHH`
  A 1-gram model just keeps track of the raw probability of performing each action and predicts based on this probability $P(\texttt{L}) = 7/15, P(\texttt{H}) = 8/15$
  Almost 50/50 in this case
  A 2-gram model keeps track of the probability of performing an action given the previous action

	L	H	
L	2/7	5/7	`LHHLHLLLHHLHLHH`
H	4/7	3/7	`LHHLHLLLHHLHLHH`

- optimal window size, hierarchical version
- application in combat games: predicting where players will be, what weapons they will use, or how they will attack. $\rightsquigarrow$ even too good
- model used in natural language processing
  probability of a word appearing given the last N-1 words

Algorithm:

- Keep track of frequencies rather than probabilities
  Use a hashtable to map prefix ($N-1$ actions) to a data record containing total number of times this prefix has been seen and hashtable of actions to counts

- Everytime you see an action
  Use the previous n-1 actions as a key to look up data record
  Create data record if this is the first time you've seen this prefix
  Update total count for prefix and the count for the nth action

- When you want to predict
  Look up prefix in hashtable
  Iterate over nested hashtable to find highest count action

Outline

Decision Learning

Given:

- behavior options to choose from
  eg, steering behaviors, animations, or high-level strategies
- observable values from the game level
  eg, distance to nearest enemy, amount of ammo left, relative size of each player's army

Desiderata:

- Learn to associate decisions with values

- Strong supervision (supervised learning): set of correct answers (maybe AI learns from humans playing)
- Weak supervision (reinforcement learning): some feedback on how good action choices are

– naive Bayes
– decision tree learning
– reinforcement learning
– neural networks

Naive Bayes

- Use Bayes theorem to determine the probability of a hypothesis being true given the current state of the world.
- "Naive" refers to the assumption that joint probabilities are conditionally independent.
- In game contexts, one could use Naïve Bayes to learn what action to take given a state of the world, or to classify game activity
- Typically used offline

- $\Pr(C \mid E) = \frac{\Pr(E|C)\Pr(C)}{\Pr(E)} = \alpha \Pr(E \mid C)\Pr(C)$

Brake?	Distance	speed
Y	Near	Slow
Y	Near	Fast
N	Far	Fast
Y	Far	Fast
N	Near	Slow
Y	Far	Slow
Y	Near	Fast

E:=Break; C:=Distance, Speed

- We can directly observe $\Pr(E \mid C)$ and $\Pr(E)$ $\Pr(C)$
- Use Bayes' theorem to compute $\Pr(C \mid E)$

$$\Pr(B \mid D, S) = \alpha \Pr(D, S \mid B) \Pr(B)$$
$$= \alpha \Pr(D \mid B) \Pr(S \mid B) \Pr(B) \quad \text{(assumption of independence)}$$

Brake, or not, in a situation where the distance to a corner is 79.2 and its speed is 12.1.
Calculate the conditional probability that a human player would brake in the same situation and use that to make our decision.

$$P(B = Y \mid D = 79.2, S = 12.1) = ? \qquad P(B = N \mid D = 79.2, S = 12.1) = ?$$

$$\Pr(B = Y \mid D = far, S = slow) = \alpha \Pr(D = f \mid B = Y) \Pr(S = s \mid B = Y) \Pr(B =$$
$$= \alpha \frac{2}{5} \frac{2}{5} \frac{5}{7} = \alpha \frac{4}{35}$$
$$\Pr(B = N \mid D = far, S = slow) = \alpha \frac{1}{14}$$

Maximum a posteriori

$$
\begin{aligned}
c_{MAP} &= \operatorname{argmax}_{c \in C} \Pr(C = c \mid E) \\
&= \operatorname{argmax}_{c \in C} \frac{\Pr(E \mid C = c) \Pr(C = c)}{\Pr(E)} \\
&= \operatorname{argmax}_{c \in C} \Pr(E \mid C = c) \Pr(C = c)
\end{aligned}
$$

$\rightsquigarrow$ use logarithms instead!

- Joint conditional probabilities require a lot of data to learn
  Number of possible instances times number of possible hypothesis

- The naïve Bayes classifier assumes that attribute values are conditionally independent.

- The probability of observing a conjunction of attributes values is the product of the probabilities for individual attributes.

$$\Pr(A_1, A_2, \ldots, A_n \mid H) = \prod_i \Pr(A_i \mid H)$$

Learning Decision Trees

- ID3 algorithm "Inductive Decision tree algorithm 3" or "Iterative Dichotomizer 3"
- Split data in one attribute such that the information gain is maximized.
- Information gain = reduction of overall entropy
- Entropy measure of information in a set of examples (the degree to which the actions in an example set agree with each other)

Healthy	In Cover	With Ammo	Attack
Hurt	In Cover	With Ammo	Attack
Healthy	In Cover	Empty	Defend
Hurt	In Cover	Empty	Defend
Hurt	Exposed	With Ammo	Defend

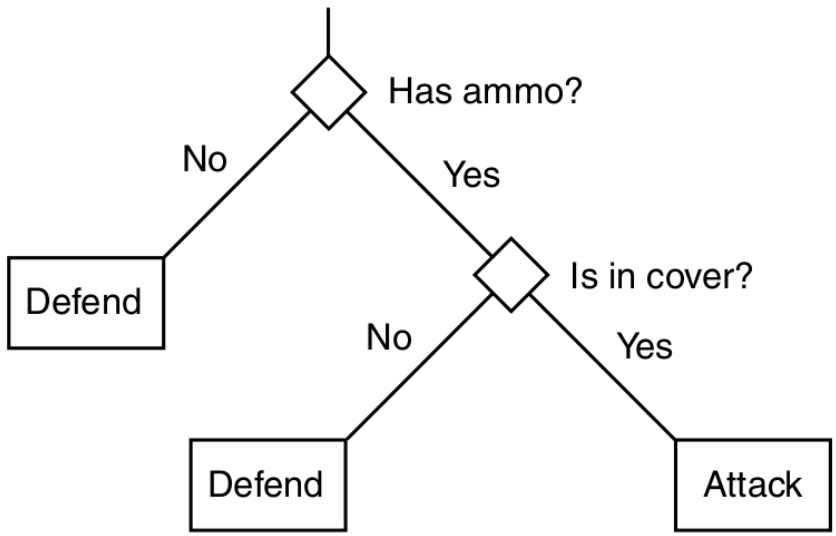
$$E = -p_A \log_2 p_A - p_D \log_2 p_D = 0.971$$

Divided by:

Health	$E_{\text{healthy}} = 1.000$	$E_{\text{hurt}} = 0.918$
Cover	$E_{\text{cover}} = 1.000$	$E_{\text{exposed}} = 0.000$
Ammo	$E_{\text{ammo}} = 0.918$	$E_{\text{empty}} = 0.000$

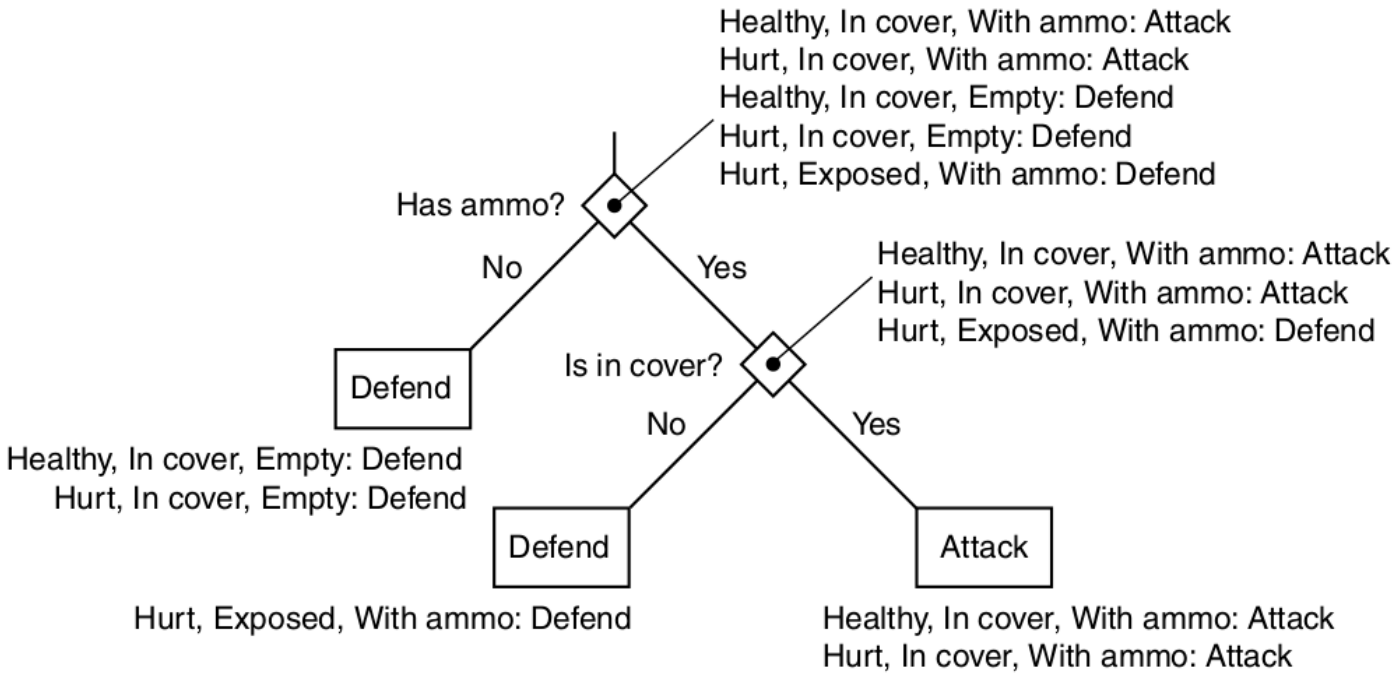
$$Gain = E - p_{left} E_{left} - p_{right} E_{right}$$

$$G_{health} = 0.020$$
$$G_{cover} = 0.171$$
$$G_{ammo} = 0.420$$


Has ammo?
No
Yes
Defend
Is in cover?
No
Yes
Defend
Attack

Other issues

- more than Two Actions, Non-Binary Discrete Attributes
- continuous attributes
- incremental Decision Tree Learning
  take the new example and use its observations to walk through the decision tree.
  - if terminal node and match then nothing needed
  - if terminal node and not match then convert node into a decision node.
  - if not terminal node, then recompute split:
    - if same split then continue the descent
    - if different split then recompute all subtree from there


Healthy, In cover, With ammo: Attack
Hurt, In cover, With ammo: Attack
Healthy, In cover, Empty: Defend
Hurt, In cover, Empty: Defend
Hurt, Exposed, With ammo: Defend
Has ammo?
No
Yes
Healthy, In cover, With ammo: Attack
Hurt, In cover, With ammo: Attack
Hurt, Exposed, With ammo: Defend
Defend
Is in cover?
Healthy, In cover, Empty: Defend
Hurt, In cover, Empty: Defend
No
Yes
Defend
Attack
Hurt, Exposed, With ammo: Defend
Healthy, In cover, With ammo: Attack
Hurt, In cover, With ammo: Attack

Reinforcement Learning

The quality of an action isn't clear at the time the action is made

Three components:

- an exploration strategy for trying out different actions in the game
- a reinforcement function that gives feedback on how good each action is
- a learning rule that links the two together.

Q-Learning

Representation of the World

- states are made up of many factors: position, proximity of the enemy, health level (state number)
- Q-learning is model-free algorithm because it doesn't try to build a model of how the world works.
- needs to know which actions are available
- feedback or reinforcement value commonly in the range $[-1, +1]$ and can be zero if no info
- an action can give different feedbacks
- carrying out an action from a state may not lead always to the same state
- $\langle s, a, r, s' \rangle$ experience tuple: start state, action taken, reinforcement value, and resulting state

Doing learning

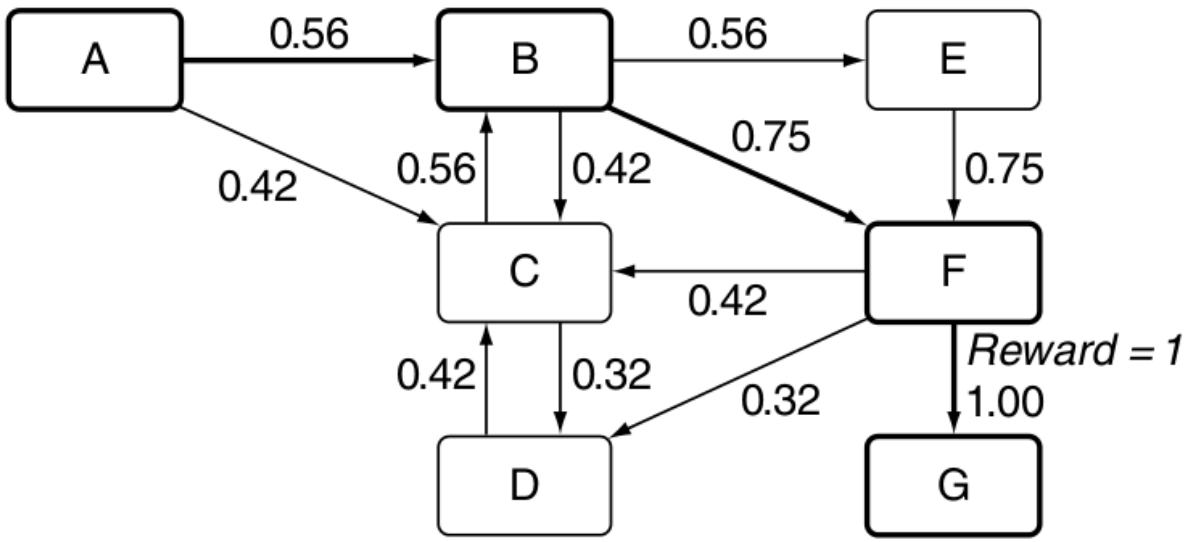
- keeps quality information Q-value for states and actions
- update:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(r + \gamma max_{a'}\{Q(s',a')\}\right),$$

α is the learning rate, and γ is the discount rate
r reinforcement value
$\gamma max_{a'}\{Q(s',a')\}$ brings the success (i.e., the Q-value) of a later action back to earlier actions

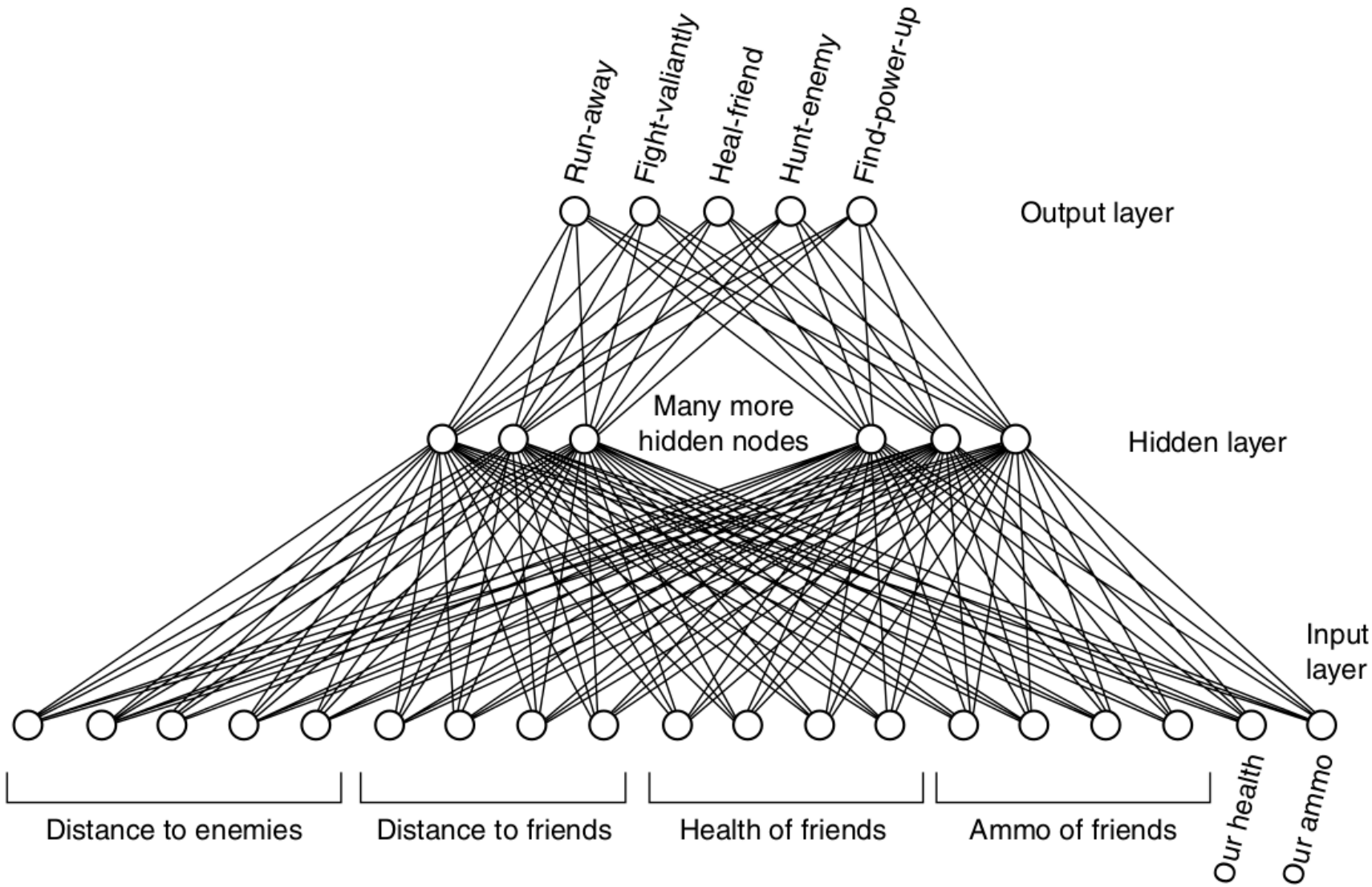
exploration strategy

- policy for selecting which actions to take in any given state.
- most commonly: select the action with the highest Q-value from the current state or select a random action.
- acting under this influence before algorithm has completed


A
B
E
C
F
D
G
0.56
0.56
0.42
0.56
0.42
0.75
0.75
0.42
0.42
0.32
0.32
Reward = 1
1.00

Artificial Neural Networks

- classification tasks
- group a set of input values (such as distances to enemies, health values for friendly units, or ammo levels) together so that we can act differently for each group
- decision trees could also work


Run-away
Fight-valiantly
Heal-friend
Hunt-enemy
Find-power-up
Output layer
Many more
hidden nodes
Hidden layer
Input
layer
Distance to enemies
Distance to friends
Health of friends
Ammo of friends
Our health
Our ammo

Outline

Evolutionary Computation

Applications of EC to develop better games by:

- Design of NPC
- Level design

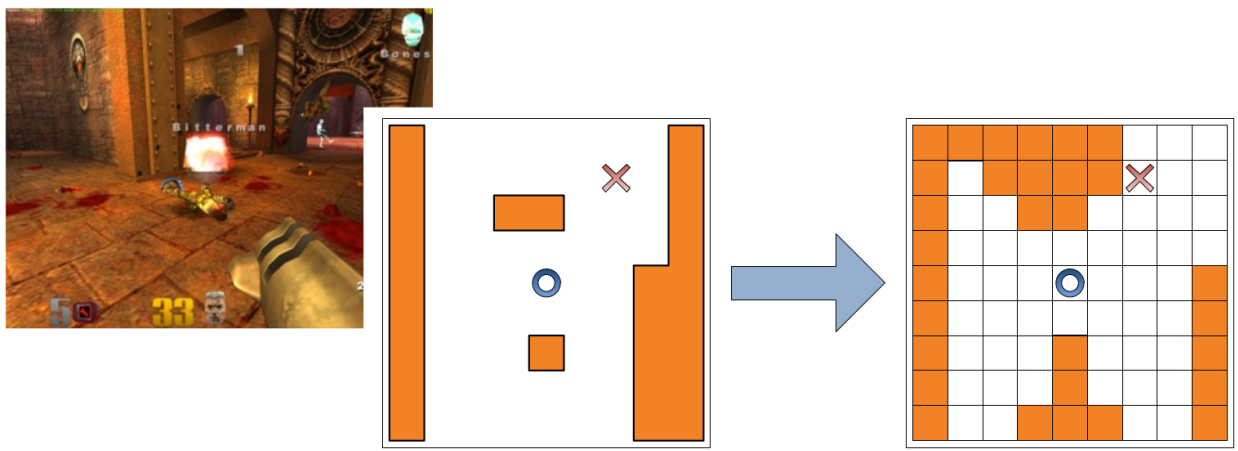
Evolutionary Design of NPC

- Early works in the field focused on beating the game...
- ... now focus is more on non-player characters (NPC), i.e., characters not controlled by the player (either opponents or an allies)
- Design choices
  How to represent the NPC?
  How compute fitness?
  Which evolutionary techniques?
- Some examples
  Evolving Quake III bot
  Evolving Racing Lines in Games

Evolving bots: representation

How to represent the game environment?

- Collect information with raycasting - Discretize local area around the NPC



Evolving bots: representation

- How to represent an NPC strategy?
  - population of if-then rules
  - game environment is matched against the rules
  - rule with the closest matching is applied

Approach

How to find the best rules?

Real-players data used to build a rule-base

- Individuals are generated by selecting a random set of rules from the rule-base
- GA is applied to evolve the best set of rules
- Recombination works on the sets of rules
- Mutation works on the single rules
- Fitness is computed as

fitness = damage dealt - damage received