The Question

Can an intelligent system design a computer game?
Computer Game

- **Mechanics***
  - rules governing logical state changes in game
- **Content**
  - non-procedural assets
  - art, sounds, animations, levels, maps etc.
Content

MUSIC  ART  ANIMATIONS  LEVELS

MAPS
By automatically designing game mechanics

- Create games unique to each player
- Generate solutions to design problems humans may not be able to think of
- Create games across domains (domain independent)
- Create new game genres
Mechanic Generation

- Create mechanics
  - from low-level primitives
  - check and update variables

- De novo synthesis of game agent actions
  - given knowledge of game domain
De Novo synthesis

Game Agent Actions → Mechanic Generation
Mechanics

- Take the form of planning operators
- Representation specialized to game mechanics
Generate and Test Process

- Generate mechanics that meet design requirements
  - Constraint Solver
- Test generated mechanics to meet playability requirements
  - Planner
  - Mechanic Generation creates the operators
Design Requirements

How actions work in a game
Playability Requirements

- Ensures you can play the game and reach some goal
- May be limits on the actions taken to achieve the goal
Process

Constraint Solver

Generate Mechanics

Planner

Test Mechanics
Mechanic Generation

Requirements -> Game Mechanics
Constraint Solver

- Generic approach to search combinatorial spaces
- Hard and soft requirements
  - Required
  - Optimized
- Searching for mechanics that meet design criteria
Planner

- Used for proving presence or absence of play traces within game domain

- Play trace: sequences of mechanic choices and state updates

Figure 1: Platformer level showing a playtrace using a generated mechanic set. Arrows indicate generated mechanics, dotted arrows indicate gravity.
Mechanic Structure

- Based on planning domain representations
- Find a sequence of operations that transform the world from initial state to one in which goal situation holds
Components of a PDDL planning task:

• Objects: Things in the world that interest us.
• Predicates: Properties of objects that we are interested in; can be true or false.
• Initial state: The state of the world that we start in.
• Goal specification: Things that we want to be true.
• Actions/Operators: Ways of changing the state of the world.

Example: Gripper Task

Room 1

Robot

Room 2

Goal: Move all the objects to Room 2
PDDL Representation

Defining the Mechanic Generation Problem

Some definitions...
Mechanic Generation

Constructing a set of game mechanics that meet playability and design requirements
Playability Requirements

- Ensures you can play the game and reach some goal
- May be limits on the actions taken to achieve the goal
Design Requirements

How actions work in a game
Game Domain

State and Transition Models
State Model

Entities that make up the game
Parameters
Allowed ranges of values
Transition Model

How states change from one another
Game Instance

- Setting from a game domain
- example: level in platformer or single battle in RPG
Avatar-Centric Mechanics

Transitions initiated by the player or other agents in the process of controlling an avatar.
Combine

Game instance for a state model with a transition model including avatar-centric mechanics yields “playable” game experience
Definitions in Action: An Example
Definitions in Action: An Example

Health + Magic (Resources)
Definitions in Action: An Example

Each “instance” has different values for resources

Player

Enemy
Definitions in Action: An Example

Playability

Requirement:

Over all instances, player can kill enemy w/o being killed
Definitions in Action: An Example

Design Requirement:

All spells have a cost
Given these requirements:
What spells (or mechanics) should be in the game?
Solution to Mechanic Generation

- State and Transition Model Representation
- Process to search for Transition models within state model
- Process to Test Transition Model (provided set of instances)
Constraint Solver is used to search for transition models that meet design criteria
Planner is used to validate transition models against playability criteria
Game Domains Representation

- Subset of PDDL ideas with extensions
- Simplified to turn-based domains
- Deterministic actions
State Model

Defines a domain of game entities in terms of their allowed states

- Entities
- Parameters
- Parameter Ranges
State Model

Entity (e)
Parameter (p)
Has(e, p)
AbsRange(p,e,r)
RelRange(p,e,r)
Example

Entity(Player)
Parameter(Health)
Parameter(Magic)
Has(Player, Health) Has(Player, Magic)
AbsRange(Health, Player, [0,1000])
AbsRange(Magic, Player, [0, 100])
Transition Model

- A set of mechanics define Transition Model
- Drawn from PDDL’s action schemas
Mechanic Model

<mechanic id, set of preconditions, set of effects>

Mechanic similar to action in PDDL
Mechanic Model: Pre and Effect

- Time-indexing
- Coordinate Frames of Reference
Time Indexing

- Preconditions can reference state at times other than present
- Effects can reference states other than next

Delayed effects or Historical State Checks
Coord Frame of Reference

- Distinguish between
  - Traditional world state terms
  - Perceived avatar relative terms
  - Preconditions and Effects relative to avatar

- Adjacency
Preconditions and Effects

<frame, time, condition>
Condition

\[ F(\text{parameter(entity), value}) \]

F is logical function
Example Mechanic: Spell

<DamageOverTime,
{(Absolute, 0, GreaterThan(Health(Enemy), 0))},
{Relative, 1, Update(Health(Enemy), -1)),
(Relative, 2, Update(Health(Enemy), -1))}}
Example Mechanic: Spell

- Check that enemy is alive
- Reduce enemy health by 1 on the next two turns
Mechanic Recombination

- One mechanic references another mechanic having occurred
Performed

Double jump

(Absolute, -1, Equal(Performed(Jump), Player))
Mechanic Generation

- Creates mechanics by choosing preconditions and effects
- Ensure conform to design requirements
- Test mechanics adhere to playability requirements
- Repeat until hard and soft requirements met
Mechanic Generation

- Expensive process
- Small game domains
- Many games use relatively small sets of mechanics?
Playability Checking

- Planner can simulate game instances using player choices among generated mechanics
Playability Requirements

1. Goals
   a. Target situation

2. Maintenance Goals
   a. begin true, must hold throughout
   b. Goals must uphold maintenance goals

3. Engine Constraints
   a. Non-avatar rules
Example

Goal: Kill all enemies
Maintenance: Be alive
Constraint: Player cannot drop below 0 magic
Examples

Role Playing
Platformer
Role Playing

- Goal: Kill all enemies
- Maintenance: Be alive
- Constraint: No negative magic
- All actions have cost
Instance

Magic  Health

Player

Enemies
Role Playing

- Example Spell: Damage All
- They assigned human readable names
- Reduce enemy health by 1, reduce player mana by 2
Figure 1: Platformer level showing a playtrace using a generated mechanic set. Arrows indicate generated mechanics, dotted arrows indicate gravity.
Platformer

- Jump
- Double Jump
- Forward
- Moves player position
Conclusions

- Key is representation
- Focus higher level problems such as generating mechanics, rather than genre specific
- More sophisticated playability requirements
Future Work

● Data?
● Performance?
● Generated game mechanics with actual users?
Thank you!

Questions?
Mechanic Adaptation

- Instead of from scratch, start w/ set of mechanics, produce minimally changed set
- Iterative design, Adjusted to meet new insights about game
Answer Set Programming

- Used to implement the constraint solver and planner
Multi-level

- Gradually introduce new mechanics over seq of levels
- Augmented with level index
- Enforces progression
Multi-Agent

- Competing agents
- Not true adversarial interaction
Controls

- Map input buttons to generated mechanics
- “Automated control generation”