Research Summary

The research focuses on AI for games. We create Sumobot, a 2D, top-down unity-based sumo robot battle intended to encourage players to learn the very basics of programming and artificial intelligence (AI). Inspired by real Sumobot competition and Battlesnake, we want to make this game as simple and as open as possible. In Sumobot, a player controls a robot, and competes against another opponent by pushing the other opponent's robot out of the arena. The arena is circle-shaped, and there is a limited timer of 60 seconds for each round. The game will be played as best-of-5 rounds. There are 3 player modes: PvP, PvAI, and AIvAI. There are 2 action turn modes: real-time and simultaneous turn-based (STB). There are three control modes: UI buttons, live code, and AI script. All winning converts to XP or coins, and can be used in the in-game market. The items can be mixed to produce new items (fusion).

Now, I try to distinguish the whole research into 3 research that can be published in the SCI journal.

First paper: Improving Dynamic Pacing-based Challenge in Sumobot.
We implement the core gameplay of Sumobot, creating several modes, and want to create the challenge according to the intended game pacing. Pacing may consist of 3 aspects: threat, movement impetus, and tempo (from the paper Pacing-based Dungeon Level Generator). But likely we will focus on threat and tempo, because movement impetus is hard to quantify. Otherwise, you can give us an example. **Contributions** in this paper are: we will implement a few ai methods to provide challenges by creating enemy ai, while also embed the concept of game pacing to the method to create a more dynamic experience.

Second paper: PCG Experimentation to Enhance Replayability in Sumobot through player designed pacing.
The second paper we focus on creating procedural modules for avatar (skin, dialog), environment (arena/ level, crowds), FX (damage, smoke). We will also create features like: mixing or hybriding marketable inventory items (visual & stats), level editor. By hybriding the item, it might impact the resulting game pacing. Hence, implicitly, players may change the game pacing by hybriding the items that can lead their gameplay with their designed/ intended pacing. **Contribution**: Implementing procedural modules on sumobot to enhance replayability, and allowing players to experiment using hybridized items to manipulate the game pacing.

Third paper: Strategy modelling in Sumobot.
In this paper, we want to explore deeper about game pacing in a smallest scale of action (single action), because in the previous thesis paper, we computed the pacing value of a room. However, it is not real-time accurate. Therefore, we want to achieve the possibility of visualising the game pacing from real-time actions. Additionally, we can also define the possible branching of the pacing. For now, let's just show the branch if the possible branches have more than 0.25 of pacing values differences. Lastly, we therefore can also do strategy modelling by showing possible action trees that are associated with the points in the pacing line (graph). So, strategy is actually a tree of actions that can also evolve through the game pacing. How can we do that? we need to find the patterns (robot action, and pacing pattern) from the recorded data. Then, we

can start to model the strategy using trees and pacing graphs. **Contributions**: Pacing as strategy modelling representation. Proposing that strategy is evolving a chained tree of actions influenced by game dynamic (pacing).

Critiques

*Notes: These are advice or recommendations.

Paper 1: Improving Dynamic Pacing-based Challenge in Sumobot

Strengths:

- **Innovative Integration:** Embedding pacing dimensions (especially threat and tempo) directly into the AI challenge mechanism is novel.
- **Practical Relevance:** Focusing on dynamic pacing that adapts to in-game conditions can significantly enhance player engagement.

Critiques:

- **Operationalization of Metrics:** Clearly define and justify how “threat” and “tempo” are quantified. Benchmarking these metrics against established dynamic difficulty adjustment systems—such as the AI Director in *Left 4 Dead* or systems studied in IEEE Transactions on Games—would provide a concrete reference point. For instance, comparing response curves or intensity thresholds against these benchmarks can help validate that the algorithm adapts challenge levels appropriately.
- **Validation and Benchmarking:** Include experiments that compare the dynamic pacing system to a fixed or static pacing baseline. The proposed dynamic pacing algorithm should be benchmarked against a static or non-adaptive baseline. Key metrics might include reaction time (how quickly the algorithm adjusts to changes in gameplay), mean session duration, and player satisfaction indices. Utilizing simulation scenarios similar to those in existing dynamic pacing studies can provide comparative data that underscores the algorithm’s effectiveness.
- **Scalability & Robustness Benchmarks:**
  It's crucial to test the algorithm across a range of game conditions. Benchmarking its performance in scenarios with varying levels of action density (e.g., comparing low-intensity versus high-intensity engagements) will reveal if the pacing adjustments hold up under diverse conditions.

Paper 2: PCG Experimentation to Enhance Replayability through Player-Designed Pacing

Strengths:

- **Procedural Diversity:** Combining avatar customization, environmental generation, and FX with an item fusion system can offer a rich, user-driven experience.
- **Player Agency:** Allowing players to indirectly manipulate pacing by hybridizing items is a creative twist that could boost replayability.

Critiques:

- **Measurement of Replayability and Consistency:** Define quantitative benchmarks for replayability—such as repeat play frequency, session length, and user engagement scores. Compare your results against baseline metrics found in studies on PCG in games like *No Man's Sky* or *Super Mario Bros.* level generators or established GAN-based PCG frameworks. Metrics such as variance in design parameters and playability scores (obtained via user studies) would serve as useful benchmarks.
- **Generation Speed and Efficiency:**
  Benchmark the time required for content generation and the computational cost against standard PCG methods. This helps determine if the new modules not only produce engaging content but do so efficiently. Comparing these metrics with those in related research on PCG in games (e.g., studies in Computers in Entertainment) can validate improvements in system performance.
- **Impact on Game Pacing:**
  Since the PCG system is designed to indirectly alter game pacing, it should be evaluated against established pacing benchmarks. For instance, measuring the change in average gameplay speed or the frequency of pacing shifts—using tools common in dynamic difficulty adjustment research—can validate that the player-designed hybridization effectively influences replayability.

Paper 3: Strategy Modelling in Sumobot

Strengths:

- **Data-Driven Insights:** Recording real-time actions and corresponding pacing data to generate action trees is an ambitious approach that can yield valuable insights into in-game strategy evolution.
- **Visualization of Dynamics:** Real-time pacing graphs and branching trees could serve as effective tools for both game designers and players.

Critiques:

- **Predictive Accuracy Benchmarks:**
  The method for modeling strategy as an action tree tied to pacing must be benchmarked for predictive accuracy. Compare your algorithm's predictions with actual gameplay outcomes using standard metrics like precision, recall, and F1 score. Benchmarks from previous work on search-based procedural content generation or decision tree models in game AI can serve as references.
- **Threshold Sensitivity Analysis:**
  The choice of a 0.25 pacing value difference for branch generation should be justified by sensitivity analysis. Benchmarking the performance of various threshold levels on synthetic and real gameplay data will help determine if this value is optimal, or if adjustments yield better alignment with strategic decision-making.
- **Comparative Analysis Against Baselines:**
  Benchmark your strategy modeling approach against simpler baseline models (e.g.,

standard decision trees or heuristic-based methods) to show the added value of your complex action tree methodology. Using controlled simulations and standard game AI metrics from prior studies in IEEE Transactions on Computational Intelligence and AI in Games will strengthen your claims.

Approximate Timeline

Paper 1: Improving Dynamic Pacing-based Challenge in Sumobot

- **Months 1-4:**
    - *Literature Review & Conceptual Framework:* Deep dive into dynamic difficulty and pacing models.
    - *Define Metrics:* Operationalize “threat” and “tempo” with clear definitions and baseline metrics.
    - *Initial Prototype:* Develop basic AI modules in Unity that adjust pacing based on these metrics.
    - *Core Gameplay Integration:* Implement various game modes (PvP, PvAI, AIvAI) with dynamic pacing features.
    - *Preliminary Playtesting:* Run initial tests with small user groups and log pacing data.
- **Months 5-7:**
    - *Iterative Refinement:* Adjust AI algorithms based on playtest feedback; refine pacing metrics.
    - *Controlled Experiments:* Design experiments to compare traditional static challenges versus dynamic pacing.
- **Months 7-9:**
    - *Data Analysis & Validation:* Analyze user feedback and gameplay data statistically.
    - *Manuscript Preparation:* Write, peer-review, and prepare the paper for submission.

Paper 2: PCG Experimentation to Enhance Replayability

- **Months 1-6:**
    - *Literature Review & Design:* Investigate PCG techniques for avatars, environments, FX, and inventory fusion.
    - *System Design:* Draft the architecture for procedural modules and player-driven item hybridization.
    - *Development Phase I:* Implement initial procedural modules in Unity (avatar skins, dialog, arena layouts, FX).
    - *Prototype Item Fusion:* Develop a basic system for mixing inventory items that affect gameplay pacing.
    - *Integration & Playtesting:* Integrate the PCG modules into Sumobot; conduct user studies to measure replayability and pacing changes.
    - *Iterative Tuning:* Adjust systems based on user data and qualitative feedback.
- **Months 7-9:**
    - *Final Data Analysis:* Compare replayability metrics (e.g., session lengths, repeat play rates) with and without PCG interventions.
    - *Paper Drafting:* Compile findings, draft the manuscript, and prepare for journal

submission.

Paper 3: Strategy Modelling in Sumobot

- **Months 1-4:**
  - *Literature Review & Methodology:* Review research on strategy modeling, action trees, and real-time pacing analysis.
  - *Define Data Collection Protocols:* Set up robust logging of in-game actions and pacing metrics.
  - *Data Collection:* Record extensive gameplay data across different modes and control schemes.
  - *Initial Analysis:* Start analyzing the relationship between individual actions and pacing changes.
- **Months 4-6:**
  - *Model Development:* Use pattern recognition and machine learning techniques to derive action trees and identify branching points (using the 0.25 threshold as a hypothesis to test).
  - *Visualization Tools:* Develop graphs and visualization tools to represent the evolving strategy.
- **Months 7-9:**
  - *Validation & Refinement:* Validate the strategy models through simulation and predictive testing.
  - *Manuscript Preparation:* Finalize analysis, document results, and draft the paper for submission.

SCI Journals for Submission

- **For Paper 1:**
  - *IEEE Transactions on Games* – Focuses on game design, AI, and dynamic systems.
  - *Entertainment Computing* (Elsevier) – Publishes research on game mechanics and adaptive gameplay.
  - *Simulation & Gaming* – Offers insights on dynamic challenge and simulation-based gameplay.
- **For Paper 2:**
  - *Entertainment Computing* – Suitable for research on procedural content generation and player-driven design.
  - ~~*Computers in Entertainment* – Focuses on innovative game design and interactive media.~~ Now Game: Research and Practice.
  - *IEEE Transactions on Games* – If the focus leans heavily on technical AI and PCG implementations.
- **For Paper 3:**
  - *IEEE Transactions on Computational Intelligence and AI in Games* – Appropriate for data-driven strategy modeling and dynamic gameplay analytics.
  - *IEEE Transactions on Affective Computing* – If the work incorporates elements of real-time player experience and emotional dynamics.
  - *Simulation & Gaming* – For research on simulation-based strategy and game mechanics modeling.

Example of Sumobot Networking Implementation

1. Use a Web-Based API for AI Bots

- Each sumobot AI should be a **separate web service** that players can write in any language.
- Unity WebGL should act as the **game engine and referee**, making API calls to the AI servers.

2. Define the API Structure

Your game should send **HTTP requests** to the player's AI server and receive movement decisions. Example API structure:

- **GET /** → Basic info about the AI.
- **POST /start** → Called at the beginning of a match.

POST /move → Called every frame or turn, expecting a move response like:
json
CopyEdit
```
{
  "move": "forward"
}
```

-
- **POST /end** → Called when the match ends.

3. Implement API Calls in Unity

- Use Unity's `UnityWebRequest` to send and receive AI moves.
- Each sumobot should have a **controller script** that makes API calls and updates movement.

Example Unity C# code for requesting AI moves:

csharp
CopyEdit
```
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;

public class SumoBotController : MonoBehaviour
{
    private string aiUrl = "http://player-ai-server.com/move";
    void Start()
    {
        StartCoroutine(GetMove());
    }

    IEnumerator GetMove()
    {
        UnityWebRequest request = UnityWebRequest.Post(aiUrl, "{}");
        yield return request.SendWebRequest();

        if (!request.isNetworkError && !request.isHttpError)
        {
            string response = request.downloadHandler.text;
            ProcessMove(response);
        }
    }

    void ProcessMove(string json)
    {
        // Parse JSON and apply movement logic
    }
}
```

4. Hosting AI Bots

- Players can **host their AIs** on cloud services (AWS, Replit, Heroku) or run them locally.
- The AI logic only needs to **handle HTTP requests** and respond with valid moves.

5. Game Loop & Physics

- Unity **simulates physics** for sumobot battles.
- AI responses **only dictate movement** (e.g., forward, turn left, push).
- The **game engine resolves physics interactions**, determining winners.

6. Scalability Considerations

- For multiplayer, use **WebSockets** or **HTTP polling** to update AI moves efficiently.
- A **match server** could manage battles and synchronize AI turns.

This setup allows **any programming language** to be used for AI while keeping Unity as the main simulation engine. Do you want help with a specific part, like setting up the Unity API calls or designing the sumobot physics?