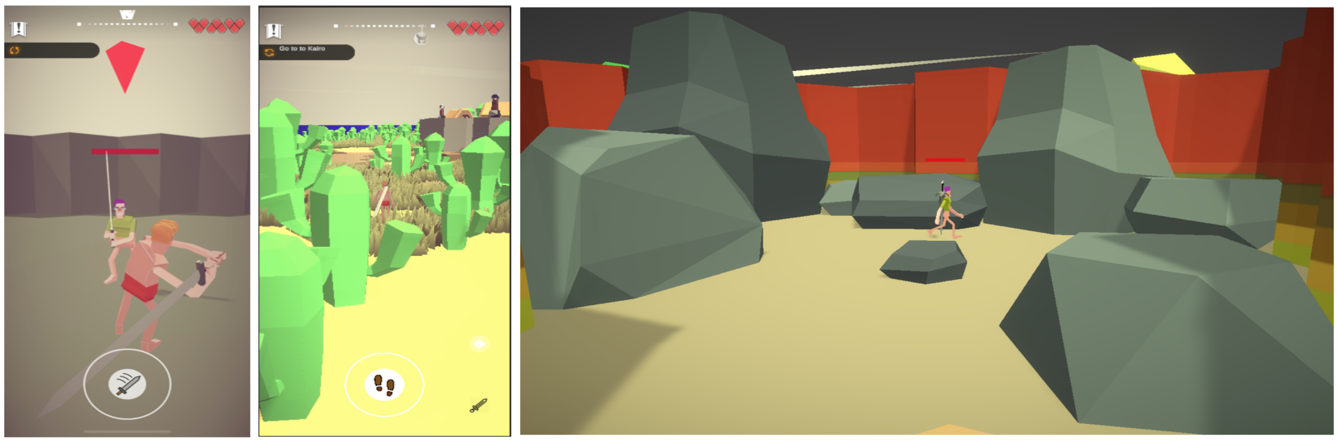
Intertwining World and Narrative Generation for a Mobile Role-playing Game

Joel Jonasson[1], **Brenden Lech**[2], **Sasha Azad**[2], **Chris Martens**[2]
[1] Blast Bit Enterprises AB, Sweden
[2] North Carolina State University, USA
joel.jonasson@blastbit.net,{bglech, sasha.azad, mawellni}@ncsu.edu, crmarten@ncsu.edu



Making a game where the environments, characters, and story are all procedurally generated is a complex task. We are a small studio in Sweden trying to develop such a game for the mobile market (working title "Lance a Lot"). Our game adopts role-playing game conventions, such as an explorable world map, non-player characters, and quests. It is our aim to develop as our "story engine" a system capable of procedurally generating quests and narratives to guide the gameplay experience. Such a system would encourage replayability and ensure players never run out of content to experience, while affording players more agency to shape the narrative and the world through their actions.

Our approach to knitting together multiple kinds of procedural generation involves two types of generators: abstract and concrete. The abstract generators construct representations of game content as sentences in first-order logic, and the concrete generators that instantiates the in-game representations of those logical sentences.

The abstract generators include generating the history of the world, cultures and belief systems, character personality and backstory. When the player enters a previously unvisited area, the engine generates the next area (see Fig. 1), beginning with the generation of the terrain, including the area's biome, preliminary information about connecting areas, and any other high-level terrain definition information required.

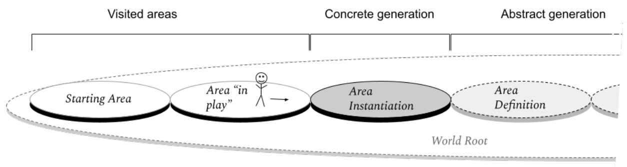


Figure 1: A root area is an area with the world root as its parent. When the player enters a new root area all content inside is expected to have been instantiated into the game world.

Our abstract generators receive as input possible player interactions, facts about the world and terrain, details available in the knowledge base, and hyper-parameters such as a series of morphological functions (Propp 2010), pacing or story beats (Mateas and Stern 2005), and other thematic variables.

The abstract generators work in a gradual process of making information more concrete until the properties of an area are defined to the degree that the information can be used as input to the concrete generators. An example of some of the properties associated with regions are locative properties (such as resources, or settlements), the non-player characters in the region (including details of their species, or interactions available to them), and quest information for the player. The first iteration will employ a partially planningbased approach to quest generation.

Each concrete (content) generator (see Fig. 2) defines a category of objects and thereby also what properties and relations may be specified for an object of that category. For example, a content generator for vegetation may define a plant as a type of content, with properties like edible and poisonous defined as unary predicates (for instance, $edible(plant)$), and defining locations as binary predicates (for example via, $at(plant, area)$). Similarly, the agent generator controls available interactions in the world, for instance, $avoids(agent, agent)$, $dazed(agent)$, or $initiates(agent, agent, Fight|Dialogue)$. These content generators are registered with the narrative system, enabling the abstract generators to see them as object categories.

```
public interface IArea : IGenObj {
    List<AreaType> Types { get; }
    public List<IArea> Children { get; }
    public List<IArea> Parent { get; }
}

public interface IWorldGenerator :
    IGenerator<IArea> {
    void WorldRoot(IArea area);
    void Sea(IArea area);
    void Forest(IArea area);
    void Desert(IArea area);
    void Village(IArea area);
    void PartOf(IArea child, IArea parent);
    void BridgePart(IArea area);
    void Bridge(IArea area);
}
```

Figure 2: An example of interface code (in C#) between a concrete generator and the abstract generators. A fact can be added by calling the matching predicate method.

```
world_root(world).

sea(a1_sea).
forest(a1_forest).
village(a1_village).

part_of(a1_sea, world).
part_of(a1_forest, a1_sea).
part_of(a1_village, a1_forest).
```

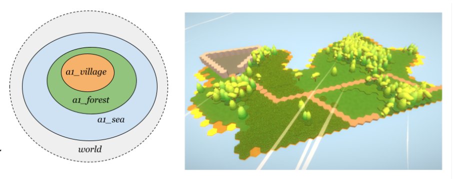


Figure 3: An example of how the generator from Fig. 2 might be used to construct a definition of the world, a simplified sketch of the result and how it looks once instantiated into the game world.

Currently we are in the process of iterating on game mechanics and improving the content generators in the game. We are actively collaborating with an academic group to design a cohesive system that will be underpinned by Answer Set Programming, Linear Logic, and/or AI planning techniques. We are interested in sharing our current approach with the EXAG community and receiving feedback on our initial prototypes.

```
world_root(world).

sea(a1_sea).
forest(a1_forest).
village(a1_village).
bridge_part(a1_bp).

part_of(a1_sea, world).
part_of(a1_forest, a1_sea).
part_of(a1_village, a1_forest).
part_of(a1_bp, a1_forest).

sea(a2_sea).
desert(a2_desert).
village(a2_village).
bridge_part(a2_bp).

part_of(a2_sea, world).
part_of(a2_desert, a2_sea).
part_of(a2_village, a2_desert).
part_of(a2_bp, a2_desert).

bridge(connection).
part_of(a1_bp, connection).
part_of(a2_bp, connection).
```

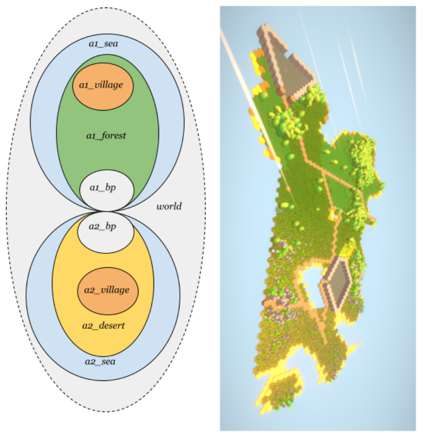


Figure 4: An expansion on Fig. 3 with two connected areas. Please note that the image on the right is somewhat misleading as it does not reflect that the two areas are connected via specific bridge parts.

References

Mateas, M., and Stern, A. 2005. Structuring content in the façade interactive drama architecture. In *AIIDE*, 93–98.

Propp, V. 2010. *Morphology of the Folktale*, volume 9. University of Texas Press.