Story Engine

Parsing Component

Responsible for building a story from a StoryTags textual document.

Logical Progression and Feedback

Holds all logical rules determining progression of a story. References static data of the Point, Reward and Item available in game. Contains in game audio data.

Visual Presentation

Presents the user interface such as animations and various feedbacks. Knows nothing about the logic behind it. Presents audio feedback. **May pose limitations on the validity of textual content, mainly length – should not be so.**

The UI represents processes made by the Engine, and acts as a controller between the user and engine. The user interacts with a button, a function is invoked at the engine. The return value is presented along with any modifications that should be presented.

Because stories are written by external users of the system, they (stories) must conform to structures supported by the game. We define this structure as an XML schema used by writers to create stories. We chose this approach for the following reasons:

* A story must be in valid structure so the game engine may play it correctly. It must conform to varying restrictions. To name a few:
  + Length of paragraph
  + Length of missing pieces in a paragraph
  + Maximum number of paragraphs in a story
  + A set of logical rules determined by the gameplay – such as “at least one correct answer in every paragraph”.
* An independent tag or script scheme supports implementation of the game engine in various technologies. The prototype shall be written in c#, but to reach a larger audience it should eventually be implemented in Html and JavaScript.
* The schema eases the development and testing of the game engine. The parsing mechanism may be altered while the overall game structure is set. In case future features introduce more game modes or additional functionality, we keep track of the schema version and support backward compatibility. To do this we need only introduce new parsing mechanisms.

These restrictions derive from the rules set by the logical and graphical game engine. For one, length of paragraph holds may be mainly restricted by the graphical view. On the other hand, the maximum number of missing pieces may be limited by the gameplay (to define feedback based on 12 steps of answering an answer i.e. “combo”.

The first draft of the schema is named *StoryTags.* It supports linear plots only. The following example presents a story, “hello world”, with one chapter, “hello you” written in StoryTags. The single chapter holds a single paragraph with one missing piece “is”. The solution to the paragraph describes one false option, “are”. Stories are the dynamic content of the Prodigy game.

Note: While we may not explicitly gather stories from users in this schema, the data shall be stored as such. However for the time being, this shall be the default story input mechanism.

<story>

<name>hello world</name>

<description>a short tail</description>

<chapter>

<name>hello you</name>

<description>first steps</description>

<paragraphs>

<paragraph>

<body>

My name <piece>is</ piece > guy.

</body>

<solution>

<option istrue=” false”>

<piece>are</piece>

</option>

</solution>

</paragraph>

</paragraphs>

</chapter>

</story>

More on this later.

NOTE: Return delta Game State to minimize bandwidth

Gameplay is focused on individual puzzles within a game. A **Game** is defined as a set of paragraph puzzles. A game ends in any of the following conditions:

* User health has completely depleted
* The user exited the game by demand
* The last puzzle in the set was completed.

Currently the only mode we support is story mode, which is a single player game mode. In this mode, the game is the entire list of paragraph puzzles in a user created Story. Story mode provides a checkpoint mechanism, where checkpoint are scattered throughout paragraphs.

In story mode, a Game is a chapter. Progression is saved at the end of each chapter, and at various checkpoints throughout the puzzles of a chapter.

Select piece from box.

Place piece in puzzle.

Combine one or more pieces and place in puzzle at once.

Remove piece from puzzle.

Undo any operation made.

Game Entities

A game contains multiple levels, each level holds one Paragraph and one Challenge max. in a challenge, a player must submit one or more individual Tokens to complete it. A sequence of Tokens matching every missing Token from a challenge is referred as a Token Permutation. You may regard it as a complete answer to a question.

Every Permutation is assigned a score by its writer. The containing challenge holds a number indicating the minimal score threshold required to successfully pass it. Thus at least one Permutation in each challenge must be assigned a score

* Paragraph Body – plain text represents the question at hand.
* Solution – current answer provided by the player and its result
* Multiple Answers – the answers available to submission by players
* Combo (game mode dependent) – current streak of consecutive individual answers within a paragraph.
* Health (game mode dependent)
* Score

As part of the requirements, the server handles all game logic while the client is responsible for rendering the UI. The server must validate all actions performed by a player and reply with the corresponding game state data. The game state holds the UI entities to draw and their values. Because multiple game modes are available, we wish to be able to provide different UI representations to each mode, where a game state in each may hold different data.

Game Modes

1. Story Mode
2. Arcade
3. Time Attack