Prodigy (working title)

Requirements

Data Dictionary

Exercise Data

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Name** | **Type** | **Range** | **Description** |
| Title | String | Max 50 |  |
| Description | string | Max 200 |  |
| Score Threshold | int | 0-100 | Determines the minimal score to pass the exercise |

Missing Content Paragraph Data

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Name** | **Type** | **Range** | **Description** |
| Complete Textual Content | String | 1-300 |  |
| Incomplete Textual Content | string | 1-300 |  |
| Solution | int | 1-5 |  |
| Parts in Answer | int | 0-12 | Determines the minimal score to pass the exercise |
| Answer Part | string | Max 50 |  |
| AnswerPart Location | int |  | Location in textual content |

Game State

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Name** | **Type** | **Range** | **Description** |
| Name | String | Max 50 |  |
| Description | string | Max 200 |  |
| Score Threshold | int | 0-100 | Determines the minimal score to pass the exercise |

World Data

|  |  |  |  |
| --- | --- | --- | --- |
| **Input Name** | **Type** | **Range** | **Description** |
| Score | int | [0,100] | An integer indicating the weight of a question clause in the entire exercise. |
| Achievement Levels / Score Thresholds | int | 2-5 | Defined by the creator, these levels describe the scores a user may receive, a mapped string indicating a name for the level, and are used to unlock content upon completion of quests. The number of different achievement levels ranges between 2 and y (do be determined later). Each one matches a percentage of questions to be completed successfully in an exercise. For example, Angry Birds uses a three level mechanism, indicated by stars to receive. The minimum number of stars required to complete a stage is one, and the Perfect Score is 3 stars. |

General Story

Client requires an online system to manage practical content presented in class. This content must be solvable and automatically checked online, without supervision of its editor.

The content to practice is in multiple answer format, which eases automatic checking and allows various languages and characters to be validated without specialized parsers.

A virtual world (classroom) is required to hold all students that are registered to the same course. A classroom defines one syllabus, which is an ordered list of subjects (named Levels), dictating progression of students. (A beginning and end date of the syllabus is not specifically required, but should be taken into consideration.)

Progression through the syllabus is done through accomplishing quests, which are practical Exercises to be submitted by students. A Quest holds a single exercise, and includes conditions for a successful completion. It also describes unlocked content available after completing the quest. Unlockable content are additional quests and the solution to the completed quest. Quests may be required or optional, but only required quests may unlock additional ones.

Game Entities and Objects

XP – experience points granted upon completion of achievements.

Game Play

A game consists of questions, traversed one at a time. With each question, a number of empty clauses must be completed with one of the available answers to the question (or clause).

Game Modes

There are to be a number of different game modes to be played. They all consist of the same game data; the Exercise.

Quest

Quest mode is a single player game, where the player must pass predefined conditions in order to accomplish the quest successfully, which results in unlocking one or more quests. Additional preconditions may be introduced that upon completion, bonus content and/or items are unlocked.

The player traverses the questions linearly, where the order of questions is defined by the writer of the exercise. He must submit answers to all clauses in a question in order to progress to the next one.

The game ends when any one of the following conditions is satisfied:

* Player health has depleted (failed to pass minimal score threshold before all questions were answered)
* All questions submitted

At any time, the user may exit the quest. In this case, progression is saved and restored upon next play.

To unlock the next quests that follow, the player must satisfy one or more preconditions, defined by the writer. The default precondition for accomplishing a quest (which is actually completing it) is the minimal score threshold one must achieve.

1. Quest Completion Precondition (Required): Achieve a score of at least X. (where

X is one of the achievement levels determined by the world admin. For example, say we assign our world 3 levels in which a quest may be completed. The first level, which is the lowest one, determines the minimal percentage of correct answers we have submitted throughout the entire exercise. The third one indicates we have achieved a perfect score, i.e. we have submitted all answers and all were correct. The second one (or any other one in case we have more than three) lies between the upper and lower bounds.

Let us take the achievement levels in the game Angry Birds as an example. Angry Birds uses a three level mechanism, indicated by Stars one receives. The minimum number of stars required to complete a stage is one (achievement level 1), and the Perfect Score is 3 stars.in our case, the following scenario relates the concept of achievement levels to the student:

During a quest, the player is notified whether the current answer to be submitted shall grant him an achievement. Say the student has successfully completed 55 percent of the exercise, and is now confronted with a question clause that will grant him 5-6 percent if completed successfully. If he is successful in answering the current clause, he shall receive a “star”. The next time a similar scenario happens in the same quest, he shall receive an additional “star” is he is successful. Gathering one star grants unlocks the next quest (if available). Gathering all three unlocks the solution. Two stars may just supply additional motivation in pursuing a perfect score.

The second default precondition, unlocks the solution of the exercise. Students who complete the quest and satisfy this precondition, receive the complete solution, as written by the Teacher. It includes all possible answers to all questions, and may include additional explanations to each.

1. Solution Unlock Precondition (Required, Default): Achieve a Perfect Score.

Additional preconditions may be imposed and assigned, either to supply an additional constraint to precondition 1, or provide unlocks of additional user created material, such as bonus quests. Other constraints may relate to the next accomplishment criteria:

* Time

A Teacher may define a maximal amount of time in which an exercise must be completed

The basic data stored about each client is the following:

* Location: a number indicating the current empty clause in the client’s solution.
* Answer: the answer submitted to the current location
* Health: the health left in percent.
* Combo: current combo multiplier level of player.
* Magic Item: magical item used.

Every action performed by a client may hold any number of properties (>1). Some actions do not include data to fill all properties.

Domain Requirements

Student: Manage the Story Book

A Student must be registered in order to play stories. Once registered, he is supplied with an account, which includes a username, password, and a bookmarking feature to retain references to stories. Stories are only playable through a **Story Book**, which is the bookmarking feature mentioned above, and must be added to one’s own for play. Stories are located in the global **Library**. To search for stories, we supply a search mechanism, where stories may be fetched by any textual content that they hold (PI). Stories are found, added to the Story Book, and then played through it.

The following actions are possible from a Story Book:

1. Play a story.
2. Remove a Story from the storybook(PI).
3. Review a finished Story (PI).
4. Review a Story’s Writer (PI).

Playing a Story from a Story Book (1) includes:

* Resume a Story from last checkpoint (or possibly from any saved checkpoint)
* Replay a Story from the start

We thoroughly explain the actual gameplay in the following section.

A teacher requires a platform where he may write playable exercises for students. Also required is a tracking mechanism which supplies a means to follow-up on students progression and interact with them. **Note: the current business model dictates that this feature shall only be available to system administrator users, not Teachers.**

Teacher: Manage the Writer’s Desk

Requirements relating to exercise creation (feature to support)

1. Define weight of each question in exercise.
2. Define the appearance of solution options in a question. If the answer is made of more than one part, should options be presented as the total of unique parts for the entire question1? As unique parts per part of answer2? As a combination of multiple parts3?
3. Define weight of individual parts of a question if more than one is described.
4. Define minimal passing threshold of the exercise.
5. Define minimal passing threshold per paragraph if answer is made of more than one part.
6. Solution options are sometimes shared by all paragraphs in an exercise – support this feature (e.g. matching).
7. Paragraphs with no solution – to present passages in story or contextual introduction to following paragraphs.
8. Paragraphs that are ‘Bonus’ material (with weight?).
9. Define the next paragraph after a solution is submitted to the previous one in the context of a single exercise.

Requirements regarding course management:

1. Easily find an exercise by any of its defining attributes:
   1. Name
   2. Writer details
   3. Lesson/subject
2. Allow dynamic addition of lessons, exercises and additional course resources.
3. Define a syllabus of lessons that are to be traversed linearly. Support dynamic addition of lessons to the syllabus.
4. Define lesson constraints to be met in order to progress to next lesson.
5. Define minimal criteria to be met in order to successfully complete a submitted exercise.
6. Define Unlockable content upon successful completion of an exercise/level.
7. Support additional resources that may be added to a class.

A teacher must also register to the system before use. Once registered, he is granted a profile which holds his username, password, and access to the **Writer’s Desk**. The latter provides means for managing all exercises created, which includes the following:

1. Manage The Story Books. A writer has multiple story books, one book for each Language that the system supports. Each story book holds all stories (pending publication, published, and unfinished). The same functionality that a **student’s** story book supplies is available to a writer (limitations may apply). This includes the *creation* and *modification* of all Stories a Teacher holds in his Writer’s Desk. A writer may request to cancel publication of a story if it is in a pending state. If a writer wants to remove a published story he must file a request through the system. Deletion of stories is available only if a story is in neither of the two latter cases.
2. Manage individual paragraphs. To support efficient capabilities for presenting a paragraph in more than one exercise. This includes:
   1. Searching for a paragraph by any of its unique features.
   2. Inserting a copy or a reference (requires unique ID) of an existing paragraph into multiple exercises.
   3. Removing all references of to a paragraph.
   4. Modifying a paragraph (includes modification of referenced paragraphs).
3. Publish a Story. When a story is considered complete by its Writer, it is must be authorized by an Admin for it to be available for students. Much like in real life, it must be checked for appropriate content and censored if required.

**UC: Create Story**

A teacher determines a subject of practice relating to some lesson taught in class. He continues to fill the following details:

* **Name** – the name of the story. Required.
* **Instructions** – the guidance required to fill answer all following questions. Required.
* **Paragraphs** – ordered appearance of individual paragraphs in the story the user is required to complete/answer in order. For each paragraph:
  1. Enter text and mark missing part using <NULL> and </NULL> tags indicating start and end of part, respectively.
  2. Now a solution is created with X parts that were tagged.
  3. If a teacher requires there to be more than one solution, creates another solution template, holding X missing parts, and enters a sequence of parts that constitute another solution. A part may be empty but the order of each solution must be different. Id the teacher requests to add “dummy” solutions, he repeats the previous process, but indicating that the solution is false (by marking a field for example).

Teacher submits paragraph for validation. If the total amount of text in the paragraph, once complete with an answer, exceeds some threshold, notify. If the total amount of unique parts in the solution exceeds some threshold, notify. Otherwise, an OK is returned, and the teacher proceeds to the next one.

* **Grade** – Once all paragraphs have been submitted, a default equal weight is assigned to all paragraphs. A user may define different weights for individual paragraphs (must be system validated, max =100%), which will determine the “health” a player has in game.
* **Grade Threshold** – usually accompanying a Grade is a minimal passing threshold. This is used to determine the minimal level of perception a student must have in order to allow some sort of progression.

The Users

The main users of the system are students of a second language (**Student**) and their teachers/tutors (**Writer**). A **Publisher** is responsible for authorizing contents of exercises before publication. **Administrator** is the person managing the system as a whole and has the highest level of Requirements relating to game derive from us, Game Designers. The following table describes all users of the system and the features required by them:

PI – Pending Implementation. Means that implementation of a certain feature or component might not be included in the course of the project.

Use Case: Play Story

Actor: Prodigy Student

Play Story: User selects a story from his **Story Book**. The Story Book contains various stories that were added from **Prodigy Library**. Stories may be ones that have yet to be played, or ones that are currently in progress. If it is one he has previously started and saved his progress, the following scenarios are possible:

1. Resume from last checkpoint. Checkpoints are scattered throughout the paragraphs of a story, and every one saves the exact current game state of the user. The beginning of every chapter (before the first paragraph) is also a checkpoint which is automatically saved. This includes:

* Health (mistakes left to make)
* Points earned until checkpoint in current chapter.
* State of magical items (if used or not).

1. Replay a previously played chapter. Replaying a chapter is made available once one has been completed, and it is played separately from the current story in progress. We shall consider making this option available only once a story has been completed.

Playing a Story:

1. Player receives first paragraph in story (question) and is presented with optional solutions to the question. If the solution is divided into separate parts, the user is required to go through all of them. In other words:
   1. For each paragraph in story
      1. For each part in solution of paragraph
         1. Submit an answer
      2. if solution is complete, reward player X points and continue to next paragraph.

The number of mistakes a user is allowed to make is determined by the

Requirements

Teacher

Student

1. **Play stories**
2. Register as a student to one or more teachers (stakeholder teacher)
3. Be assigned exercises from a teacher (stakeholder)
4. Review exercises
5. Review teachers

Game Designer

1. Allow player to submit a solution in the following manners:
   1. To an entire exercise
   2. To a paragraph in an exercise
   3. To a part in a paragraph.

**Submitting a solution means the player receives feedback and is not able to modify it later.**

1. Allow player to undo a submission in the following manners:
   1. To an entire exercise
   2. To a paragraph in an exercise
   3. To a part in a paragraph.
2. Provide clues and assistance in an exercise. A clue may refer to one or more parts of answer to a paragraph (1), one or more paragraphs in an exercise (2).
   1. Reveal the size (in case of multiple solutions) of the part (number of characters) if relevant (concerning exercise formats). (1).
   2. Eliminate one or more parts from the options. (1) and (2).
   3. Reveal the number of missing parts if not indicated. (2).
   4. Place a solution and query the user for its validity.
3. Provide a solution to an exercise in the following ways:
   1. **Selection from available options.**
   2. *Free text (?)*
4. Indicate the score a user received in an exercise based of the grading scheme of the writer.
5. Provide different levels of difficulty a user may play at regarding the gameplay of exercises and not the actual content.
6. *Create different User interfaces for different formats of exercises. Obsolete.*
7. Consider future implementations of multiplayer online gaming.
8. *Support online and offline single player mode. Obsolete.*
9. Online single player mode should be considered in the architecture to allow future versions to be playable only online, such as is the case with a HTML version.

Game Play Modes (controlled by creator)

1. TimeLimit:
   1. Per game
   2. Per challenge (?)
   3. Per obstacle (?)
2. Answer submission // imposes constraints on validity of data
   1. Complete answer to challenge
   2. Answer to obstacle
3. Progression (next state after move)
   1. After correct move – next move
   2. After false move
      1. Next move
      2. Same move (try again)
4. Allow/disallow use of special items
5. Show score per obstacle / per challenge / per game
   1. Per Obstacle – score of obstacle is presented to user. User knows if submitted answer was correct or not after each submitted obstacle
   2. Per challenge – user is presented with a score of the entire challenge.
      1. Also with specific scores per Obstacle
      2. No feedback on individual obstacles is returned

// FROM PREVIOUS DRAFT //

Prodigy (working title)

Analysis and Requirements

**Prodigy is an online interactive language practice game system, centered on the concept of practice through contextual stories.**

Prodigy is a game. Requirements relating to the game derive from us.

Motivation

Build an online system where teachers write and publish various exercises, and students practice them. Engulf these exercises in an enjoyable and rewarding environment for student to play with. Provide an efficient mechanism for teachers to write exercises with great freedom and ease.

Analysis

The Exercises

Exercises are the heart of the system. They are the content of individual games. They are managed by teachers (the creators of content), and engulfed in a gaming environment when presented to students.

The following steps concern the creation of an exercise as usually done by a teacher (Order is varying):

1. Select the subject of practice. The subject of an exercise is the subject of a lesson.
2. Select the preferred way to check individual exercises. Here the teacher must decide whether the set of possible answers is technically finite or not (to be reviewed later).
3. State the instructions required to fulfill the exercise. E.g. *add “a” or “an” where necessary*.
4. Until completion
   1. Enter an exercise item. This is a unique context associated with the instructions of the exercise. In case where the student has options representing possible solutions to the paragraph, they may be added here. Alternatively, they may be assigned to the exercise as a whole.
   2. Write a Solution to the previous item. Add explanations and notes if necessary. In case the student was presented with prefixed options, indicate the correct one/s.
5. Define a scoring scheme for each exercise items. A number indicating the weight an item from the total exercise may be assigned.
6. Name the exercise (optional).

Step (2) refers to the way a student submits an answer to an exercise item (Paragraph). There are two possible ways ta student may submit an answer: either he selects an item (word, sentence etc.) from a set of possibilities, or not.

The following diagram shows a UML modal of an exercise.

// TODO: EXERCISE UML MODAL HERE //

NOTE: exercise is a puzzle in the program. Having instructions and then individual paragraphs each is a puzzle.

// TODO: EXERCISE FORMAT ANALYSIS //

After analyzing a number of different exercises, we have defined the following two base templates to be sufficient in representing most types of exercises.

The three main formats of exercises present in ESL practice (but not limited to) are:

1. Fill in the blanks
2. Matching
3. Short answer questions
4. **Multiple answer questions**

The last format is marked bold because it is slightly different from the rest. The reason is that any of the above may possess options a student may select as a solution. However, the different formats ARE meant to practice different “techniques”.

Exercises are divided in two categories:

1. Exercises that contain a set of answers relevant to ALL paragraphs.
2. Ones that contain a set of answers for EACH paragraph.

We define exercises that present the user with possible solutions as having Options.

|  |  |  |  |
| --- | --- | --- | --- |
| Paragraph Format | With Options Per Paragraph | With Options for All Paragraphs | Without Options |
| Missing Content | An Option is given as a sequence of all missing parts to a single paragraph | A set of unique Parts comprises the available options a user has throughout the exercise | Free text input, the number of missing places is indicated |
| Matching | Not relevant | Same as above | Not relevant |
| Short Answer | An Option is given as one complete unit | Not relevant | Free text input. If the answer is made of more than one part, the number of parts is indicated |

Formal Definitions

Motivation

Describe a single logic used to represent and validate solutions to all types of paragraph, and allow all types of paragraphs to have more than one solution.

We formally define the modal of a paragraph. We shall use some terms from the software realm.

String – an array of one or more characters. An array of length zero represents the Null String. If not otherwise specified, refereeing to a String includes the Null String.

**Paragraph**: an array of strings of size > 0. If a paragraph holds missing content, the missing strings shall appear as Null Strings in the paragraph.

// TODO: PREDICATE HERE //

**Answer Template**: a sequence of Null Strings representing the number of individual Strings used to create a complete solution. If a paragraph holds Null Strings, indicating missing content, the size of the answer template must be equal or larger than the number of Null Strings in the paragraph.

**Missing Part**: a sequence of zero or more characters matching one or more sockets in a paragraph.

**Missing Content**: any exercise that requests a user to complete one or more absent characters from the paragraphs it holds is referred to as a *Missing Content Exercise*.

**Short Answers**: any exercise other than the above.

**Option**: an ordered permutation of one or more strings matching the answer template of a paragraph in size. May evaluate to True or False.

**Solution**: an ordered permutation of one or more strings, matching the answer template of a Paragraph in size. Evaluates to True. A paragraph may have more than one solution, where all must conform to the same size.

The following table describes the variant attributes present in an exercise

Example 1

**Format**: Fill in the blanks

**Instructions**: Use a “some” or “any”

**Paragraph***: There isn’t \_\_\_ boot-polish in this tin.*

Example

**Format**: Match

**Instructions**: match a word to its meaning

**Paragraph***: pilot*

Example 2

**Format**: Short answer questions

**Instruction**: *Put into the singular.*

**Paragraph***: Balls are toys.*

The following presents three common types of language practice exercises (examples included).

Syntax Related – practice the syntax of the language.

1. Missing Content – any exercise that requires a learner to fill missing text from an assortment of possibilities. The physical requirement is not important. If a student is required to fill missing spaces by hand or select from a range of possibilities is not an issue. The options to answer – the context of an exercise – should be available to a student. This must hold in order to tightly couple an exercise with the subject of practice.

*There isn’t \_\_\_ boot-polish in this tin.*

*Add “some” or “any”.*

1. Editable Content – any exercise that presents complete content items and requests a student to alter the content in some way.

*Balls are toys.*

*Put into the singular.*

Knowledge Related – practice knowledge of language or culture.

1. Matching – any exercise that presents complete content items and requires a student to enter an answer matching the content item.

*Black, blue, gray, table.*

*Select the odd one.*

All exercises share one base template, made of the following:

1. A Subject: name
2. A Teacher: name
3. Instructions: how to accomplish the task, received from teacher.
4. A solution: mapping between items and answers with additional (optional) explanations per answer.
5. Content Items: a set of unique items that are to be answered.
6. Answer presentation: how a student enters answers, respectful to the task.
7. Attached contextual media (optional): text, image, animation, sound.
8. Additional guidance: notes and tips.

Domain Requirements

Student: Manage the Story Book

A Student must be registered in order to play stories. Once registered, he is supplied with an account, which includes a username, password, and a bookmarking feature to retain references to stories. Stories are only playable through a **Story Book**, which is the bookmarking feature mentioned above, and must be added to one’s own for play. Stories are located in the global **Library**. To search for stories, we supply a search mechanism, where stories may be fetched by any textual content that they hold (PI). Stories are found, added to the Story Book, and then played through it.

The following actions are possible from a Story Book:

1. Play a story.
2. Remove a Story from the storybook(PI).
3. Review a finished Story (PI).
4. Review a Story’s Writer (PI).

Playing a Story from a Story Book (1) includes:

* Resume a Story from last checkpoint (or possibly from any saved checkpoint)
* Replay a Story from the start
* Replay a previously accomplished Chapter from a story

We thoroughly explain the actual gameplay in the following section.

A teacher requires a platform where he may write playable exercises for students. Teacher also require a tracking mechanism which supplies a means to follow-up on students progression and interact with them. **Note: the current business model dictates that this feature shall only be available to system administrator users, not Teachers.**

Teacher: Manage the Writer’s Desk

Requirements relating to exercise creation (feature to support)

1. Define weight of each question in exercise.
2. Define the appearance of solution options in a question. If the answer is made of more than one part, should options be presented as the total of unique parts for the entire question1? As unique parts per part of answer2? As a combination of multiple parts3?
3. Define weight of individual parts of a question if more than one is described.
4. Define minimal passing threshold of the exercise.
5. Define minimal passing threshold per paragraph if answer is made of more than one part.
6. Solution options are sometimes shared by all paragraphs in an exercise – support this feature (e.g. matching).
7. Paragraphs with no solution – to present passages in story or contextual introduction to following paragraphs.
8. Paragraphs that are ‘Bonus’ material (with weight?).

A teacher must also register to the system before use. Once registered, he is granted a profile which holds his username, password, and access to the **Writer’s Desk**. The latter provides means for managing all exercises created, which includes the following:

1. Manage The Story Books. A writer has multiple story books, one book for each Language that the system supports. Each story book holds all stories (pending publication, published, and unfinished). The same functionality that a **student’s** story book supplies is available to a writer (limitations may apply). This includes the *creation* and *modification* of all Stories a Teacher holds in his Writer’s Desk. A writer may request to cancel publication of a story if it is in a pending state. If a writer wants to remove a published story he must file a request through the system. Deletion of stories is available only if a story is in neither of the two latter cases.
2. Manage individual paragraphs. To support efficient capabilities for presenting a paragraph in more than one exercise. This includes:
   1. Searching for a paragraph by any of its unique features.
   2. Inserting a copy or a reference (requires unique ID) of an existing paragraph into multiple exercises.
   3. Removing all references of to a paragraph.
   4. Modifying a paragraph (includes modification of referenced paragraphs).
3. Publish a Story. When a story is considered complete by its Writer, it is must be authorized by an Admin for it to be available for students. Much like in real life, it must be checked for appropriate content and censored if required.

**UC: Create Story**

A teacher determines a subject of practice relating to some lesson taught in class. He continues to fill the following details:

* **Name** – the name of the story. Required.
* **Instructions** – the guidance required to fill answer all following questions. Required.
* **Paragraphs** – ordered appearance of individual paragraphs in the story the user is required to complete/answer in order. For each paragraph:
  1. Enter text and mark missing part using <NULL> and </NULL> tags indicating start and end of part, respectively.
  2. Now a solution is created with X parts that were tagged.
  3. If a teacher requires there to be more than one solution, creates another solution template, holding X missing parts, and enters a sequence of parts that constitute another solution. A part may be empty but the order of each solution must be different. Id the teacher requests to add “dummy” solutions, he repeats the previous process, but indicating that the solution is false (by marking a field for example).

Teacher submits paragraph for validation. If the total amount of text in the paragraph, once complete with an answer, exceeds some threshold, notify. If the total amount of unique parts in the solution exceeds some threshold, notify. Otherwise, an OK is returned, and the teacher proceeds to the next one.

* **Grade** – Once all paragraphs have been submitted, a default equal weight is assigned to all paragraphs. A user may define different weights for individual paragraphs (must be system validated, max =100%), which will determine the “health” a player has in game.
* **Grade Threshold** – usually accompanying a Grade is a minimal passing threshold. This is used to determine the minimal level of perception a student must have in order to allow some sort of progression.

The Users

The main users of the system are students of a second language (**Student**) and their teachers/tutors (**Writer**). A **Publisher** is responsible for authorizing contents of exercises before publication. **Administrator** is the person managing the system as a whole and has the highest level of Requirements relating to game derive from us, Game Designers. The following table describes all users of the system and the features required by them:

|  |  |
| --- | --- |
| Users | Requirements |
| Student | Relating to content of stories and gameplay |
| Teacher | Relating to managing the creation of stories |
| Publisher | Authorize story for publication |
| Publisher Admin | Override authorized stories by Publishers |
| Teacher Admin | Track story usage statistics by individual students |
| System Admin | Manage user accessibility and overall system |
| Game Designer | Features required of the game |

|  |  |  |  |
| --- | --- | --- | --- |
|  | User | Use case names | notes |
| Game Designers | Game Editor | -Manage Game rules and reward system.  -Manage StoryTags schema and parsing mechanism | Manage the logical game engine |
| Game Admin |  |  |
| Graphics Design Admin | -Manage graphical engine | Graphical engine should use data from global game rules. Needs to conform to the progression model of a story, defined by StoryTags version. |
| System Admins | System Admin | -Manage users | Get data statistics |
|  |  |  |  |
| Content Editors | Content Screener | -Review and Authorize/Reject Stories and return feedback to writer |  |
| Content Admin | -Manage languages/fields of practice |  |

Student

Teacher

Publisher

System Admin

PI – Pending Implementation. Means that implementation of a certain feature or component might not be included in the course of the project.

Use Case: Play Story

Actor: Prodigy Student

Play Story: User selects a story from his **Story Book**. The Story Book contains various stories that were added from **Prodigy Library**. Stories may be ones that have yet to be played, or ones that are currently in progress. If it is one he has previously started and saved his progress, the following scenarios are possible:

1. Resume from last checkpoint. Checkpoints are scattered throughout the paragraphs of a story, and every one saves the exact current game state of the user. The beginning of every chapter (before the first paragraph) is also a checkpoint which is automatically saved. This includes:

* Health (mistakes left to make)
* Points earned until checkpoint in current chapter.
* State of magical items (if used or not).

1. Replay a previously played chapter. Replaying a chapter is made available once one has been completed, and it is played separately from the current story in progress. We shall consider making this option available only once a story has been completed.

Playing a Story:

1. Player receives first paragraph in story (question) and is presented with optional solutions to the question. If the solution is divided into separate parts, the user is required to go through all of them. In other words:
   1. For each paragraph in story
      1. For each part in solution of paragraph
         1. Submit an answer
      2. if solution is complete, reward player X points and continue to next paragraph.

The number of mistakes a user is allowed to make is determined by the

Requirements

Teacher

Student

1. **Play stories**
2. Register as a student to one or more teachers (stakeholder teacher)
3. Be assigned exercises from a teacher (stakeholder)
4. Review exercises
5. Review teachers

Game Designer

1. Allow player to submit a solution in the following manners:
   1. *To an entire exercise. Obsolete.*
   2. To a paragraph in an exercise
   3. To an obstacle in a paragraph.

**Submitting a solution means the player receives feedback and is not able to modify it later.**

1. Allow player to undo a submission in the following manners:
   1. *To an entire exercise. Obsolete.*
   2. *To a paragraph in an exercise. Obsolete.*
   3. To one or more obstacles in a paragraph.
2. Provide clues and assistance in an exercise. A clue may refer to one or more parts of answer to a paragraph (1), one or more paragraphs in an exercise (2).
   1. Reveal the size interval (in case of multiple solutions) of the part (number of characters) if relevant (concerning exercise formats). (1).
   2. Eliminate one or more parts from the options. (1) and (2).
   3. Reveal the number of missing parts if not indicated. (2).
   4. Place a solution and query the user for its validity.
3. Provide a solution to an exercise in the following ways:
   1. Free text
   2. **Selection from available options.**
4. Indicate the score a user received in an exercise based of the grading scheme of the writer.
5. Provide different levels of difficulty a user may play at regarding the gameplay of exercises and not the actual content.
6. *Create different User interfaces for different formats of exercises. Obsolete.*
7. Consider future implementations of multiplayer online gaming.
8. Support online and offline single player mode.
9. Online single player mode should be considered in the architecture to allow future versions to be playable only online, such as is the case with a HTML version.

// END FROM PREVIOUS DRAFT //

Glossary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Term | Definition and Information | Format | Validation Rules | Aliases |
| Exercise | A composite of Paragraphs solved by the student |  |  |  |
| Paragraph | A list of strings (size>=0) making for the textual body. Strings of size 0 indicate locations where answer parts are to be inserted. | Max size with longest possible answer is 300 chars. |  | Missing Content Paragraph |
| Answer | A composite sequence of individual textual parts making a complete solution to a paragraph. An answer may be true or false. | String list. Concatenated size must be less than 100 chars. |  | Option |
| Answer Part | An individual String that is a part of an answer. Can be a Null String. | String. 0<=size<50 |  |  |
| Exercise Solution | Mapping of an answer to a paragraph |  |  | Solution |
| Null string | String of size = 0. Used to reserve place for answer parts. |  |  |  |
| Obstacle | Location in a paragraph where an answer part may be submitted to |  |  |  |
| Answer Size | the number of answer parts in an answer |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Term | Definition and Information | Format | Validation Rules | Aliases |
| Game Mode | Rules concerning the linear progression model of a game. Affects the game state |  |  |  |
| Game State | A composite of attributes concerning the state of the current game played. Consists of:  Available Moves  Available Options  Past Moves Results  Combo  Points  Special Items States |  |  |  |
| Available Submission | Paragraph number(s) + answer part number(s) that a player is able to submit answers to. |  |  |  |
| Available Option | Answer Part number(s) within a specific Paragraph that a player may choose from to submit to an Obstacle |  |  |  |
| Special Item | A rule describing how to modify the game state in favor of the player. |  |  | Magical Item |
| Combo | A rule describing how points are multiplied when correct moves are played sequentially |  |  |  |
| Special Item Threshold | The amount of points needed to allow use of a specific special item |  |  | Charge Threshold |
| Point | A number representing a score granted to a player after a correct move |  |  |  |
| Move | Submission of an option to an obstacle. Elimination of an Option to a specific option |  |  |  |

Coding Conventions

Java

Pure Block Layout

Server Side Development

(*inspired by sourceforge,net/apps/trac/svbg/wiki/Server-side%20Development*)

Design Rationale

High Priority

* There should be no UI necessary to test and develop server-side code.
* The server implements all game logic. Client inputs should be validated.
* The server should know the expected task for each client at each point.
  + Which player’s turn is it? reject input from other users (multiplayer)
  + Which task is expected by that player? Reject invalid input.
  + Provide end-user compatible error messages.

Lower Priority

* The server should know the data needs to be sent to a client. The data should be minimized, e.g. by sending incremental data.
* It should be possible to replay a complete game and/or have an overview of what happened in which turn after the game is finished.
* The server should minimize its own state.
* The server should be able to restore its state form persistent storage.
* The game logic should be implemented in a readable, extensible way. A state machine? Rules?

Nice To Have

* Undo/Redo functionality.
* An overview of how much time each player requires.

Game State Modifications

|  |  |  |
| --- | --- | --- |
| State Modification | Phase/State | Description |
| Submit Answer | Challenge |  |
| Use Item | Challenge |  |
| Move To paragraph | Paragraph | Continue to next paragraph if current state = paragraph. Move to next in current state = challenge |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Prodigy (working title)

Draft 2.0

**Prodigy is an online interactive language practice game system, centered on the concept of practice through contextual stories.**

Prodigy is a game. Requirements relating to the game derive from us.

Motivation

Build an online system where teachers write and publish various exercises, and students practice them. Engulf these exercises in an enjoyable and rewarding environment for student to play with. Provide an efficient mechanism for teachers to write exercises with great freedom and ease.

Design Requirements

1. Allow player to submit a solution in the following manners:
   1. To an entire exercise
   2. To a paragraph in an exercise
   3. To a part in a paragraph.

Submitting a solution means the player receives feedback and is not able to modify it later.

1. Allow player to undo a submission in the following manners:
   1. To an entire exercise
   2. To a paragraph in an exercise
   3. To a part in a paragraph.
2. Provide clues and assistance in an exercise. A clue may refer to one or more parts of answer to a paragraph (1), one or more paragraphs in an exercise (2).
   1. Reveal the size interval (in case of multiple solutions) of the part (number of characters) if relevant (concerning exercise formats). (1).
   2. Eliminate one or more parts from the options. (1) and (2).
   3. Reveal the number of missing parts if not indicated. (2).
   4. Place a solution and query the user for its validity.
3. Provide a solution to an exercise in the following ways:
   1. Free text
   2. Selection from available options.
4. Indicate the score a user received in an exercise based of the grading scheme of the writer.
5. Provide different levels of difficulty a user may play at regarding the gameplay of exercises and not the actual content.
6. Consider future implementations of multiplayer online gaming.
7. Support online and offline single player mode.
8. Online single player mode should be considered in the architecture to allow future versions to be playable only online, such as is the case with a HTML version.

Student: Manage the Story Book

A Student must be registered in order to play stories. Once registered, he is supplied with an account, which includes a username, password, and a bookmarking feature to retain references to stories. Stories are only playable through a **Story Book**, which is the bookmarking feature mentioned above, and must be added to one’s own for play. Stories are located in the global **Library**. To search for stories, we supply a search mechanism, where stories may be fetched by any textual content that they hold (PI). Stories are found, added to the Story Book, and then played through it.

The following actions are possible from a Story Book:

1. Play a story.
2. Remove a Story from the storybook(PI).
3. Review a finished Story (PI).
4. Review a Story’s Writer (PI).

Playing a Story from a Story Book (1) includes:

* Resume a Story from last checkpoint (or possibly from any saved checkpoint)
* Replay a Story from the start

We thoroughly explain the actual gameplay in the following section.

Use Case: Play Story

Play Story: User selects a story from his **Story Book**. The Story Book contains various stories that were added from **Prodigy Library**. Stories may be ones that have yet to be played, or ones that are currently in progress. If it is one he has previously started and saved his progress, the following scenarios are possible:

1. Resume from last checkpoint. Checkpoints are scattered throughout the paragraphs of a story, and every one saves the exact current game state of the user. This includes:

* Health (mistakes left to make)
* Points earned until checkpoint.
* State of magical items (if used or not).

**Playing a Story**

Player receives first paragraph in story (question) and is presented with optional solutions to the question. If the solution is divided into separate parts, the user is required to go through all of them. In other words:

While (EndGame == false)

For each paragraph in story

* + 1. For each part in solution of paragraph
       1. Submit an answer
    2. Submit Paragraph // or automatically submitted when last part inserted
    3. Reward player for each correct part / decrease piece from health with each mistake.

End Game

* If health is depleted
* If all paragraph finished
* If user exited game

A user may receive special items to use in game, which are basically clues:

* Clue 1 – reveal number of characters in one or more parts
* Clue 2 – remove one or more parts that evaluate false in one or more selected parts.
* Clue 3 – system selects one or more random parts as a solution and asks user if this is the correct solution or not (turns paragraph into Yes/No question).
* Clue 4 – allow undo operation of one or more parts.

A teacher requires a platform where he may write playable exercises for students. Teacher also require a tracking mechanism which supplies a means to follow-up on students progression and interact with them. **Note: the current business model dictates that this feature shall only be available to system administrator users, not Teachers.**

Teacher: Manage the Writer’s Desk

Requirements relating to exercise creation (feature to support)

1. Define weight of each question in exercise.
2. Define the appearance of solution options in a question. If the answer is made of more than one part, should options be presented as the total of unique parts for the entire question1? As unique parts per part of answer2? As a combination of multiple parts3?
3. Define weight of individual parts of a question if more than one is described.
4. Define minimal passing threshold of the exercise.
5. Define minimal passing threshold per paragraph if answer is made of more than one part.
6. Solution options are sometimes shared by all paragraphs in an exercise – support this feature (e.g. matching).
7. Paragraphs with no solution – to present passages in story or contextual introduction to following paragraphs.
8. Paragraphs that are ‘Bonus’ material (with weight?).

A teacher must also register to the system before use. Once registered, he is granted a profile which holds his username, password, and access to the **Writer’s Desk**. The latter provides means for managing all exercises created, which includes the following:

1. Manage The Story Books. A writer has multiple story books, one book for each Language that the system supports. Each story book holds all stories (pending publication, published, and unfinished). The same functionality that a **student’s** story book supplies is available to a writer (limitations may apply). This includes the *creation* and *modification* of all Stories a Teacher holds in his Writer’s Desk. A writer may request to cancel publication of a story if it is in a pending state. If a writer wants to remove a published story he must file a request through the system. Deletion of stories is available only if a story is in neither of the two latter cases.
2. Manage individual paragraphs. To support efficient capabilities for presenting a paragraph in more than one exercise. This includes:
   1. Searching for a paragraph by any of its unique features.
   2. Inserting a copy or a reference (requires unique ID) of an existing paragraph into multiple exercises.
   3. Removing all references of to a paragraph.
   4. Modifying a paragraph (includes modification of referenced paragraphs).
3. Publish a Story. When a story is considered complete by its Writer, it is must be authorized by an Admin for it to be available for students. Much like in real life, it must be checked for appropriate content and censored if required.

**UC: Create Story**

A teacher determines a subject of practice relating to some lesson taught in class. He continues to fill the following details:

* **Name** – the name of the story. Required.
* **Instructions** – the guidance required to fill answer all following questions. Required.
* **Paragraphs** – ordered appearance of individual paragraphs in the story the user is required to complete/answer in order. For each paragraph:
  1. Enter text and mark missing part using <NULL> and </NULL> tags indicating start and end of part, respectively.
  2. Now a solution is created with X parts that were tagged.
  3. If a teacher requires there to be more than one solution, creates another solution template, holding X missing parts, and enters a sequence of parts that constitute another solution. A part may be empty but the order of each solution must be different. If the teacher requests to add “dummy” solutions, he repeats the previous process, but indicates that the solution is false (by marking a field for example).

Teacher submits paragraph for validation. If the total amount of text in the paragraph, once complete with an answer, exceeds some threshold, notify. If the total amount of unique parts in the solution exceeds some threshold, notify. Otherwise, an OK is returned, and the teacher proceeds to the next one.

* **Grade** – Once all paragraphs have been submitted, a default equal weight is assigned to all paragraphs. A user may define different weights for individual paragraphs (must be system validated, max =100%), which will determine the “health” a player has in game.
* **Grade Threshold** – usually accompanying a Grade is a minimal passing threshold. This is used to determine the minimal level of perception a student must have in order to allow some sort of progression.

UC: Create New Paragraph

Precondition: story exists and paragraph amount has not exceeded.

Writer adds a new paragraph to a story and defines the body. Three cases are possible:

1. Missing Content: writer enters complete paragraph and marks missing parts using special XML tags.
2. Complete Content: writer enters textual paragraph with no missing content. If selected, missing content tags are not allowed.
3. Passageway: paragraph with no solution. Writer enters textual content.

Writer enters a solution for the paragraph:

1. A default true option is automatically created from the parts defined as missing.
2. Writer selects number of parts in an answer. Enters values in one or more parts and selects

Top level design

Database – holds Story items as XML documents. User data as RDB/FF system.

Login Server – manages access to feature in a system based on user access rights. Manages all access to data and retains state of games user play.

Desktop / HTML game engine component – holds logical rules and visual user interface.

Desktop:

* + Game Logic Component – holds state of game and determines result of user triggered moves. A Story is loaded in an instance of this component and played through it.

This plays the content of a Story in XML format.

* + Server Communication – interacts with Game logic to send progress messages to cloud. Used as checkpoint mechanism and in case some game logic that resides in the cloud must be used to determine game states.
  + GUI – represents a story in its current state. Binds to Game engine story and presents alterations in game state.

HTML:

* + JavaScript library as the game logical engine.
  + HTML5 user interface.