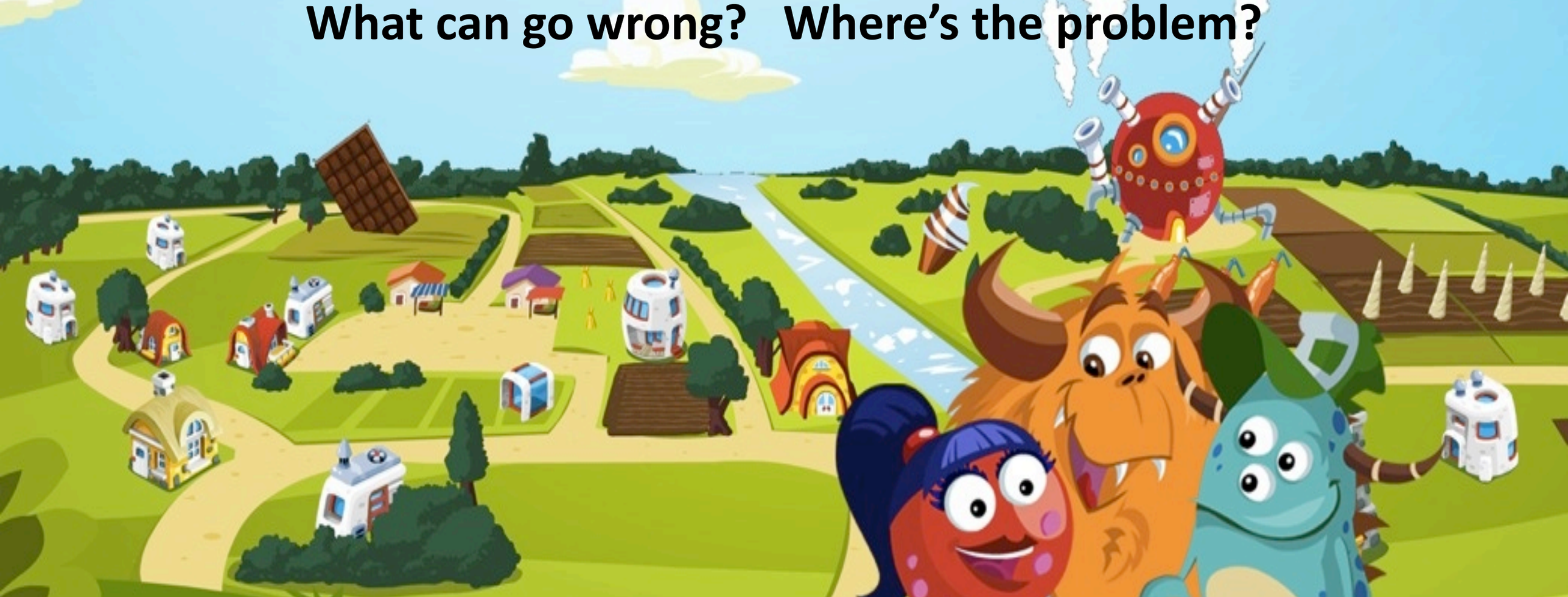
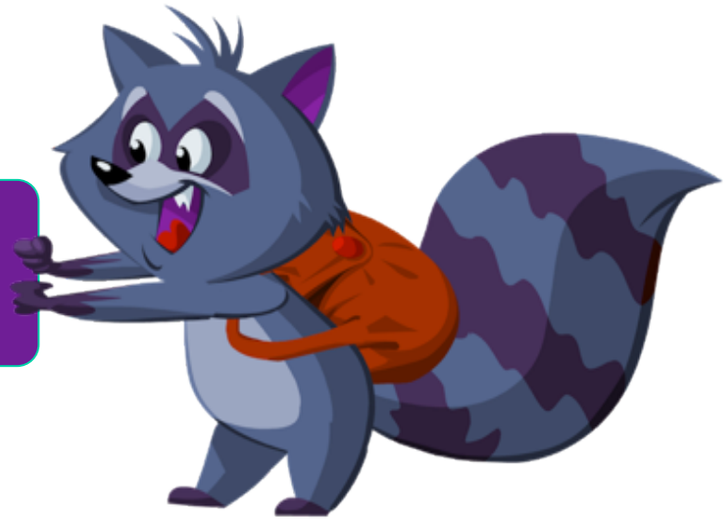
Real Time User Interactions

wooga
world of gaming

What can go wrong? Where's the problem?



Context



Examples of Game Actions

Synchronization Strategies

Context: Social Games

Social Games are

- social: play with your friends on a social network
- casual: 5 minutes are enough to play
- easy: simple to learn
- massive in scale: played by millions every day
- everywhere: can be played using a web browser

Context: Basic setup

The server is more than just a database

- Client: loads complete state from server at session start, all state changes (actions) are sent to server
- Server: keeps state in RAM, all client actions are replayed and validated, state is updated accordingly, persistence can be ignored (for now)
- The client should never wait on a server response when executing actions to ensure best usability

Context: User Interaction

Most social games use asynchronous interaction

- Player A sends a message that player B interacts with when he comes online.
- Examples are Facebook's feedposts and requests.
- Think of a message inbox that is processed when a player comes online.

Context: User Interaction

This is about Real Time User Interaction

- Player A watches player B doing something, i.e. their client display the other players actions.
- Both players are online at the same time.
- Since the game is browser based communication delays of up to 2 (even 10!) seconds are possible.
- Player actions may conflict - like two players trying to pick up the same item from the floor.

Context: Game Concepts

Some key concepts of "our" game are:

- Location: Each player has a location that she manages.

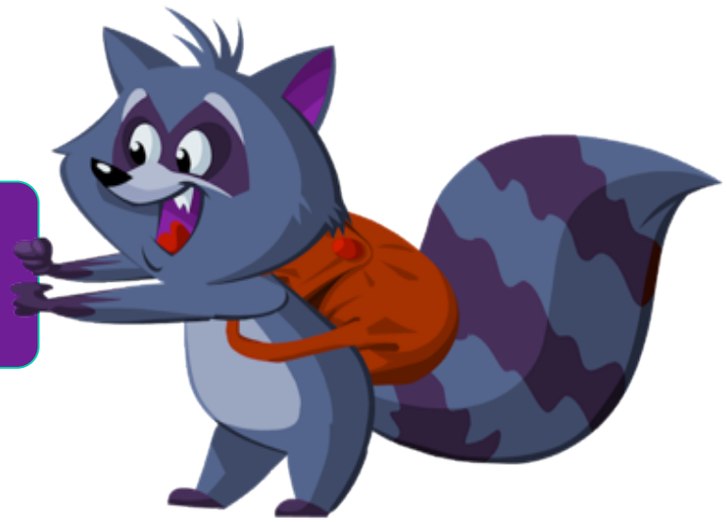
- Visits: Players can visit friends' locations: Players are either "at home" or "at a friend's location".

- Offline: After some inactivity, players become offline. This means that other player can no longer see or interact with them (they "left" the game).

Context

Examples of Game Actions

Synchronization Strategies



Examples of Game Actions

Imagine this scenario:

- Players manage their garden with a lot of fruit trees.
- Players can chop down trees, pick fruits, sell fruits, plant new trees etc.
- Players can visit other player's location and can do the same actions there as they can do in their own garden.

Examples of Game Actions

Positive example: Something we can handle easily

- Players can add stones to a stone pyramid at the bottom of a tree. The higher the pyramid the more fruits any player can pick from the tree.
- Internally this would probably be increment operations on an integer keeping track of the number of stones. Easy unless you have overflow.

Examples of Game Actions

Negative example: A conflict we cannot handle

- Player A is trying to chop down a tree while player B is trying to climb that.
- Only one action can (or should :-)) win. Either player A or B should do their action, but not both!

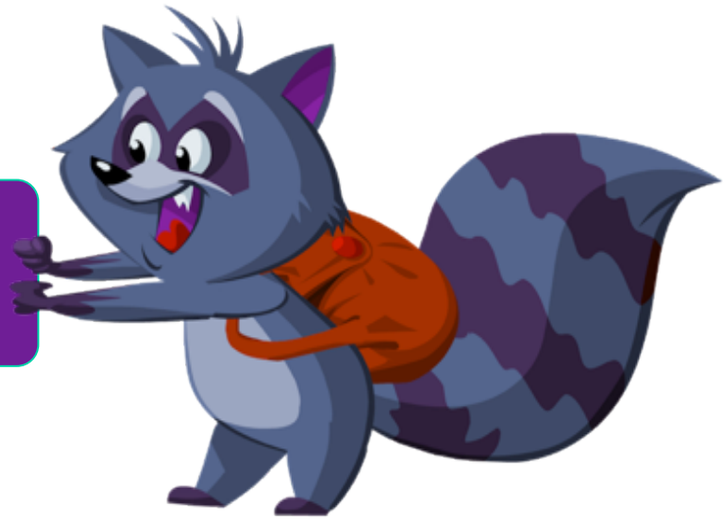
Examples of Game Actions

Indifferent example: Something we might handle

- 5 apples lie on the floor. Both player A and B try to pick up 3. Then they want to sell them.
- We might actually let both players sell 3 apples because it “feels” better from the player perspective than “rolling back” one action and thus probably failing the subsequent sell action.

Context

Examples of Game Actions



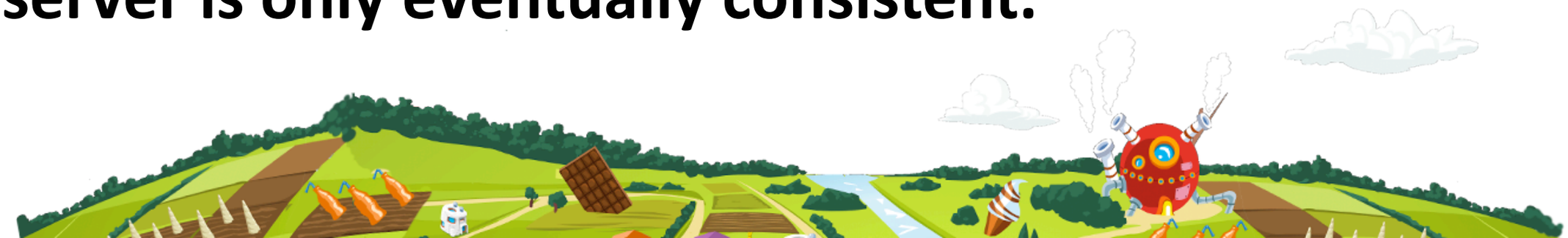
Synchronization Strategies

Synchronization Strategies

A player's client running in a browser sends http requests to a central server whenever it's game state is changed by a player action.

The game servers replays (and validate) the client action to update it's internal game state of that players.

Due to communication lags (up to 10 seconds) the internal state presentation between client(s) and server is only eventually consistent.



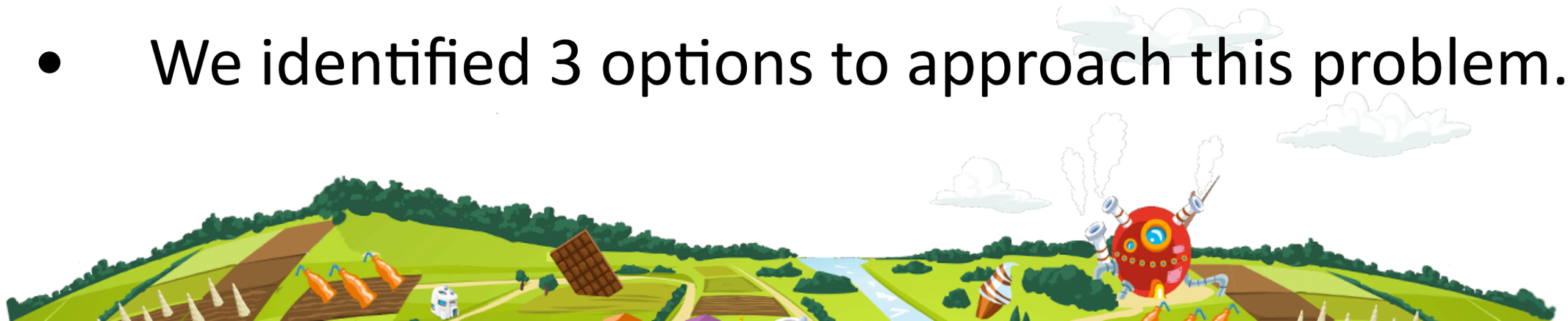
Synchronization Strategies

If two players update the same state, i.e. they are at the same location, things get more complex.

- Both clients and the server have different views on the state.

- Client actions may be conflicting, i.e. an action by one client may make the action of another one impossible or affect its outcome.

- We identified 3 options to approach this problem.



Synchronization Strategies

Option 1: Pessimistic conflict resolution

- Before a client does an action that may result in a conflict, it checks with the server if the action is valid. Very similar to 2-phase-commit approach.

- The server's state is relevant, clients have only views of the server state.

- Bad for player experience as on a click their will a second-long delay to wait for the server response.

Synchronization Strategies

Option 2: Optimistic conflict resolution

- Client executes the action and informs the server afterwards. Server then checks the action. Client may need to roll back action in case of a conflict.
- Conflict handling on the server only, client may need to rollback changes.
- Bad for player experience as player action may suddenly be rolled back.

Synchronization Strategies

Option 3: Schizophrenic conflict resolution

- When player B visits player A a copy of the game state created, so that both have their own copy.

- Own actions are simply executed and actions by other players are checked if they are conflicting. Conflicting actions are simply ignored.

- Negative: Complex and conflicts are only delayed until the next session start (but are less obvious)