METHOD AND SYSTEM TO VALIDATE IN-GAME ACTIONS IN A MULTIPLAYER ONLINE GAME

Inventors: Scott G. Miller, Austin, TX (US); Nimal Malle, Austin, TX (US); Robert Anderson, Austin, TX (US)

Filed: Sep. 26, 2011
Related U.S. Application Data
Continuation of application No. 13/215,543, filed on Aug. 23, 2011.

ABSTRACT
This disclosure generally relates to systems and methods to validate in-game actions performed in a multiplayer online game. Subsequent to performance of one or more in-game actions by a game engine that includes game code on a client-side client system, one or more corresponding validation actions may be performed by a validation engine on a server-side validation system, the validation engine having validation code that is identical to the game code. Results of execution of the validation actions may be compared to provisional game state information resulting from performance of the in-game actions, to validate the in-game actions.
Receive Action Identifiers 504

Receive Provisional Game State Information 508

Execute Validation Actions 512

Compare Results of Validation Actions to Provisional Game State Information 516

FIG. 5A

Transmit Validation Request 532

Receive Validation Result 536

Generate Updated Game State Information 540

Persist Updated Game State Information 544

FIG. 5B
METHOD AND SYSTEM TO VALIDATE IN-GAME ACTIONS IN A MULTIPLAYER ONLINE GAME

CLAIM OF PRIORITY

[0001] This application is a continuation of and claims the benefit of priority under 35 U.S.C. §120 to U.S. patent application Ser. No. 13/215,543, filed on Aug. 23, 2011, which is hereby incorporated by reference herein in its entirety.

TECHNICAL FIELD

[0002] This disclosure generally relates to games and applications in general and, in particular, to computer-implemented online games, such as online role-playing games (RPGs) that are playable by more than one person from more than one location.

BACKGROUND

[0003] In many online computer games, there is a virtual world or some other imagined playing space where a player of the game controls one or more player characters (herein “characters,” “player characters,” or “PCs”). Player characters can be considered in-game representations of the controlling player. As used herein, the terms player, user, entity, neighbor, friend, and the like may refer to the in-game player character controlled by that player, user, entity, or friend, unless context suggests otherwise. A game display can display a representation of the player character. A game engine accepts inputs from the player, determines player character actions, decides outcomes of events, and presents the player with a game display illuminating game play. In some games, there are multiple players, wherein each player controls one or more player characters.

[0004] Many online computer games are operated on an online social network. Such a network allows both users and other parties to interact with the computer games directly, whether to play the games or to retrieve game- or user-related information. Internet users may maintain one or more accounts with various service providers, including, for example, online game networking systems and online social networking systems. Online systems can typically be accessed using browser clients (e.g., Firefox, Chrome, Internet Explorer).

[0005] A client-side computing device or computer system may present the online game to the user by executing coded game logic or scripts for the online game. For example, a player may visit a virtual city of the online game, and may perform an in-game action by initiating a battle between the player’s player character and another character in the virtual city. To perform and animate the battle, the player’s client computing device may execute game view logic (e.g., JavaScript or ActionScript) to generate a visual representation of the in-game action, while execution of the battle or in-game action by game logic on the client computing device may make changes to a game state associated with the player based on the in-game action.

[0006] Such client-side execution of in-game actions allows near real-time interaction between the player and the client computing device, promoting immersive gameplay action. Unauthorized modification of game view logic and social games state information by a player on the client computing device may, however, allow devious users to obtain an unfair advantage.

DESCRIPTION OF THE DRAWINGS

[0007] FIG. 1 illustrates an example of a system for implementing particular disclosed embodiments.

[0008] FIG. 2 illustrates a more detailed view of an example system for implementing an example embodiment.

[0009] FIG. 3 illustrates an example validation system for implementing an example embodiment.

[0010] FIG. 4 illustrates an example game management system for implementing particular disclosed embodiments. FIGS. 5A and 5B illustrate high-level views of respective methods of implementing exemplary embodiments.

[0011] FIG. 6 illustrates a more detailed view of a flowchart for an example method of implementing a particular disclosed embodiment.

[0012] FIG. 7 illustrates an example data flow in a system.

[0013] FIG. 8 illustrates an example network environment.

[0014] FIG. 9 illustrates an example computer system architecture.

DESCRIPTION OF EXAMPLE EMBODIMENTS

[0015] One example embodiment may provide a method and system to validate, on a server-side validation system, in-game actions performed on a client system in a multiplayer online game by executing server-side validation actions identical to the in-game actions on server-side coded game logic identical to client-side coded game logic by which the in-game actions were initially executed. Results of server-side execution of the validation actions may be compared to results of client-side execution of the in-game actions, and the in-game actions may be validated if the respective results are identical.

[0016] The example method may thus comprise receiving one or more action identifiers that respectively indicate one or more corresponding in-game actions or steps executed on a client system or client computing device responsive to player inputs received from a player, the one or more in-game actions having been executed on the client system by coded game logic, and receiving provisional game state information with respect to a game state associated with the player after execution of the one or more in-game actions on the client system. The embodiment may further comprise executing, based on the one or more action identifiers, one or more validation actions or steps identical to the one or more in-game actions, with the validation actions being executed on a validation system remote from the client system by coded validation logic that is identical to the coded game logic on the client system. Results of execution of the one or more validation actions may be compared to the provisional game state information, to validate the one or more in-game actions. Game state information may, for example, comprise: a player character’s spatial location or orientation in a virtual environment of a game instance; a player’s score, experience score, energy level, or the like; and/or the state of an in-game environment or world associated with the player.

[0017] The method may further comprise generating verification game state information as at least part of the results of execution of the one or more validation actions, and validating the one or more in-game actions when the provisional game state information is identical to the verification game state information. Thus, in instances where the provisional game state information is identical to the verification game state information, either for a particular action or for a group or sequence of actions, the corresponding in-game action(s)
may be validated. However, if the provisional game state information is not identical to the verification game state information, the in-game action(s) may be invalidated. As used herein, the term “invalidate” includes failure of a validation operation or validation check. The method may also comprise receiving prior game state information that indicates previously validated game state information with respect to the player immediately before execution of the one or more in-game actions. In such a case, the generation of the verification game state information may be based at least in part on the prior game state information, the verification game state information being generated by applying to the prior game state information game state changes caused by execution of the one or more validation actions.

[0018] The provisional game state information and/or the verification game state information may comprise at least one game state delta that indicates one or more changes to a game state caused by performance of the one or more in-game actions. The at least one game state delta may thus comprise one or more game step deltas corresponding respectively to the one or more in-game actions, each of the one or more game step deltas indicating changes to the game state caused by performance of a corresponding in-game action. In such a case, generating of the verification game state information may comprise generating one or more verification step deltas corresponding to the respective player actions, each verification step delta indicating one or more changes to a verification game state caused by performance of a corresponding mediational action. The operation of comparing the results of execution of the validation actions to the provisional game state information may in turn comprise counting each game step delta to a corresponding verification step delta, and validating a particular in-game action if its associated verification step delta is identical to the corresponding game step delta. The in-game actions or steps may thus be validated individually or separately, even though the plurality of in-game action identifiers may be received together for validation, for example in a validation request.

[0019] The one or more in-game actions that are to be validated may comprise a sequence of in-game actions. In such a case, the method may further comprise validating the sequence of in-game actions individually and in sequence until a particular in-game action is validated, or until the entire sequence of in-game actions is validated. Validation of the one or more in-game actions may in such case comprise producing verification game state information associated with a last or latest validated in-game action in the sequence of in-game actions. Such verification game state information may be transmitted to a master game state system to persist or store the last or furthest validated game state information.

[0020] The coded game logic may be configured to perform a prerequisite check with respect to at least one of the in-game actions and/or validation actions, the prerequisite check being to establish whether predefined game state prerequisites or assumptions for executing the corresponding action or step are satisfied. The method may in such case comprise determining that a particular in-game action is invalidated if the prerequisite check for the corresponding validation action fails. In such case, the results of execution of the one or more validation actions may include an indication as to whether or not the respective validation actions are executed by the coded validation logic, e.g., whether or not the respective prerequisite checks succeeded or failed. It will be appreciated that failure of a prerequisite check, and hence invalidation of the corresponding in-game action, necessarily results in a difference between the provisional game state information and the verification game state information, because game state changes resulting from performance of the particular in-game action (and reflected in the provisional game state information) are not replicated in the verification game state information, due to nonperformance of the corresponding validation action.

[0021] An example game environment for implementing the above-described method and system is set forth below, whereafter the example embodiment is described in greater detail, in the context of the example game environment.

Example Game Environment

[0022] FIG. 1 illustrates an example of a system for implementing various disclosed embodiments. In particular embodiments, system 100 comprises player 101, social networking system 140, game networking system 150, client system 130, and network 160. The components of system 100 can be connected to each other in any suitable configuration, using any suitable type of connection. The components may be connected directly or over a network 160, which may be any suitable network. For example, one or more portions of network 160 may be an ad hoc network, an intranet, an extranet, a virtual private network (VPN), a local area network (LAN), a wireless LAN (WLAN), a wide area network (WAN), a wireless WAN (WWAN), a metropolitan area network (MAN), a portion of the Internet, a portion of the Public Switched Telephone Network (PSTN), a cellular telephone network, another type of network, or a combination of two or more such networks.

[0023] Social networking system 140 is a network-addressable computing system that can host one or more social graphs. An electronic social networking system typically operates with one or more social networking servers providing interaction between users such that a user can specify other users of the social networking system as “friends.” A collection of users and the “friend” connections between users can form a social graph that can be traversed to find second, third and more remote connections between users, much like a graph of nodes connected by edges can be traversed.

[0024] Social networking system 140 can generate, store, receive, and transmit social networking data. Social networking system 140 can be accessed by the other components of system 100 either directly or via network 160. Game networking system 150 is a network-addressable computing system that can host one or more online games. Game networking system 150 can generate, store, receive, and transmit game-related data, such as, for example, game account data, game input, game state data, and game displays. Game networking system 150 can be accessed by the other components of system 100 either directly or via network 160. Player 101 may use client system 130 to access, send data to, and receive data from social networking system 140 and game networking system 150. Client system 130 can access social networking system 140 or game networking system 150 directly, via network 160, or via, a third-party system. As an example and not by way of limitation, client system 130 may access game networking system 150 via social networking system 140. Client system 130 can be any suitable computing device, such as a personal computer, laptop, cellular phone, smartphone, computing tablet, or the like.
Although FIG. 1 illustrates a particular number of players 101, social networking systems 140, game networking systems 150, client systems 130, and networks 160, this disclosure contemplates any suitable number of players 101, social networking systems 140, game networking systems 150, client systems 130, and networks 160. As an example and not by way of limitation, system 100 may include one or more game networking systems 150 and no social networking system 140. As another example and not by way of limitation, system 100 may include a system that comprises both social networking system 140 and game networking system 150. Moreover, although FIG. 1 illustrates a particular arrangement of player 101, social networking system 140, game networking system 150, client system 130, and network 160, this disclosure contemplates any suitable arrangement of player 101, social networking system 140, game networking system 150, client system 130, and network 160.

The components of system 100 may be connected to each other using any suitable connections 110. For example, suitable connections 110 include wireline (such as, for example, Digital Subscriber Line (DSL) or Data Over Cable Service Interface Specification (DOCSIS)), wireless (such as, for example, Wi-Fi or Worldwide Interoperability for Microwave Access (WiMAX)) or optical (such as, for example, Synchronous Optical Network (SONET) or Synchronous Digital Hierarchy (SDH)) connections. In particular embodiments, one or more connections 110 each include an ad hoc network, an intranet, an extranet, a VPN, a LAN, a WLAN, a WAN, a WWAN, a MAN, a portion of the Internet, a portion of the PSTN, a cellular telephone network, another type of connection, or a combination of two or more such connections. Connections 110 need not necessarily be the same throughout system 100. One or more first connections 110 may differ in one or more respects from one or more second connections 110. Although FIG. 1 illustrates particular connections 110 between player 101, social networking system 140, game networking system 150, client system 130, and network 160, this disclosure contemplates any suitable connections between player 101, social networking system 140, game networking system 150, client system 130, and network 160. As an example and not by way of limitation, in particular embodiments, client system 130 may have a direct connection to social networking system 140 or game networking system 150, bypassing network 160.

Game Networking Systems

In an online computer game, a game engine manages the game state of the game and effects changes to the game state based on in-game actions performed by a player (e.g., player 101 of FIG. 1). A game state comprises all game play parameters, including player character state, non-player character (NPC) state, in-game object state, game world state (e.g., internal game clocks, game environment), and other game play parameters. Each player 101 controls one or more player characters (PCs). The game engine controls all other aspects of the game, including non-player characters (NPCs) and in-game objects. The game engine also manages game state, including player character state for currently active (online) and inactive (offline) players.

In the example environment illustrated in FIGS. 1 and 2, an online game can be administered by a game networking system 150, while a game engine 204 may be hosted on the client device or client system 130. The game networking system 150 can be accessed by the client system 130 using any suitable connection. A player may have a game account on game networking system 150, wherein the game account can contain a variety of information associated with the player (e.g., the player’s personal information, financial information, purchase history, player character state, game state). In some embodiments, a player may play multiple games administered by game networking system 150, which may maintain a single game account for the player with respect to all the games, or multiple individual game accounts for each game with respect to the player. In some embodiments, game networking system 150 can assign a unique identifier to each player 101 of an online game administered by game networking system 150. Game networking system 150 can determine that a player 101 is accessing the online game by reading the user’s cookies, which may be appended to Hypertext Transfer Protocol (HTTP) requests transmitted by client system 130, and/or by the player 101 logging onto the online game.

In embodiments in which the game engine 204 is provided by the client system 130, player 101 may access the game and control the game’s progress via client system 130 (e.g., by inputting commands to the game at the client device). Client system 130 may display the game interface by use of game view logic 208 (FIG. 2), receive inputs from player 101, and may perform in-game actions or events responsive to the user inputs by means of game logic 212 forming part of the game engine 204. The game logic 212 may effect changes to game state information associated with the player 101 caused by the in-game actions performed responsive to user input. The client system 130 may also maintain a game state library 216 that stores game state information indicative of the game state associated with the player 101. Game state information may, for example, include player state information and world state information.

The client system 130 may be in continuous communication with the game networking system 150 or may intermittently transfer to the game networking system 150 update information with respect to in-game actions executed by the game engine 204. Client system 130 can thus, for example, download client components of an online game, which are executed locally, while a remote game server, such as game networking system 150, provides backend support for the client components and may be responsible for maintaining the application data of the game, updating and/or synchronizing the game state based on the game logic 212 and each input from the player 101, and transmitting instructions to client system 130. Execution of the game engine 204 on the client system 130 enables off-line and/or asynchronous gameplay by a user via the client system 130.

Game Play

In particular embodiments, player 101 can engage in, or cause a player character controlled by him to engage in, one or more in-game actions. For a particular game, various types of in-game actions may be available to player 101. As an example and not by way of limitation, a player character in an online role-playing game may be able to interact with other player characters, build a virtual house, attack enemies, go on a quest, and go to a virtual store to virtual items. As another example and not by way of limitation, a player character in an online poker game may be able to play at specific tables, place bets of virtual or legal currency for certain amounts, discard or hold certain cards, play or fold certain hands, and play in an online poker tournament.
In particular embodiments, player 101 may engage in an in-game action by providing one or more user inputs to client system 130. Various actions may require various types and numbers of user inputs. Some types of in-game actions may require a single user input. As an example and not by way of limitation, player 101 may be able to harvest a virtual crop by clicking on it once with a mouse. Some types of in-game actions may require multiple user inputs. As another example and not by way of limitation, player 101 may be able to throw a virtual fireball at an in-game object by entering the following sequence on a keyboard: DOWN, DOWN and RIGHT, RIGHT, B. This disclosure contemplates engaging in in-game actions using any suitable number and type of user inputs.

In particular embodiments, player 101 can perform an in-game action on an in-game object or with respect to another player character. An in-game object is any interactive element of an online game. In-game objects may include, for example, PCs, NPCs, in-game assets and other virtual items, in-game obstacles, game elements, game features, and other in-game objects. This disclosure contemplates performing in-game actions on any suitable in-game objects. For a particular in-game object, various types of in-game actions may be available to player 101 based on the type of in-game object. As an example and not by way of limitation, if player 101 encounters a virtual bear, the game engine may give him the option of shooting the bear or petting the bear. Some in-game actions may be available for particular types of in-game objects but not other types. As an example and not by way of limitation, if player 101 encounters a virtual rock, the game engine may give him the option of moving the rock; however, unlike the virtual bear, the game engine may not allow player 101 to shoot or pet the virtual rock. Furthermore, for a particular in-game object, various types of in-game actions may be available to player 101 based on the game state of the in-game object. As an example and not by way of limitation, if player 101 encounters a virtual crop that was recently planted, the game engine may give him only the option of fertilizing the crop, but if player 101 returns to the virtual crop later when it is fully grown, the game engine may give him only the option of harvesting the crop.

In particular embodiments, the game engine may cause one or more game events to occur in the game. Game events may include, for example, a change in game state, an outcome of an engagement, a completion of an in-game obstacle, a transfer of an in-game asset or other virtual item, or a provision of access, rights, and/or benefits. In particular embodiments, a game event is any change in game state. Similarly, any change in game state may be a game event. This disclosure contemplates any suitable type of game event. As an example and not by way of limitation, the game engine may cause a game event where the virtual world cycles between daytime and nighttime every 24 hours. As another example and not by way of limitation, the game engine may cause a game event where a new instance, level, or area of the game becomes available to player 101. As yet another example and not by way of limitation, the game engine may cause a game event where player 101’s player character heals one hit point every 5 minutes.

In particular embodiments, a game event or change in game state may be an outcome of one or more in-game actions. The game engine can determine the outcome of a game event or a change in game state according to a variety of factors, such as, for example, game logic or rules, player character in-game actions, player character state, game state of one or more in-game objects, interactions of other player characters, or random calculations. As an example and not by way of limitation, player 101 may overcome an in-game obstacle and earn sufficient experience points to advance to the next level, thereby changing the game state of player 101’s player character (it advances to the next character level). As another example and not by way of limitation, player 101 may defeat a particular boss NPC in a game instance, thereby causing a game event where the game instance is completed, and the player advances to a new game instance. As yet another example and not by way of limitation, player 101 may pick the lock on a virtual door to open it, thereby changing the game state of the door (it goes from closed to open) and causing a game event (the player can access a new area of the game).
tion, the game engine may create a first game instance when a first player initially accesses an online game, and that same game instance may be loaded each time the first player accesses the game.

[0039] In particular embodiments, the set of in-game actions available to a specific player may be different in a game instance that is associated with that player (e.g., in which the player is a host player) compared to a game instance that is not associated with that player (e.g., in which the player is a guest player). The set of in-game actions available to a specific player in a game instance associated with that player may be a subset, superset, or independent of the set of in-game actions available to that player in a game instance that is not associated with him. As an example and not by way of limitation, a first player may be associated with Blackacre Farm in an online farming game. The first player may be able to plant crops on Blackacre Farm. If the first player accesses a game instance associated with another player, such as Whiteacre Farm, the game engine may not allow the first player to plant crops in that game instance. However, other in-game actions may be available to the first player, such as watering or fertilizing crops on Whiteacre Farm.

Example System

[0040] FIG. 2 illustrates a game networking system 150 for implementing particular disclosed embodiments. The game networking system 150 includes a validation system 220 to validate in-game actions performed in an online game. In the example embodiment of FIG. 2, the validation system 220 is to validate in-game actions in a multiplayer online game, typically a massively multiplayer online game, but in other embodiments, the methodologies and systems described herein can be employed to validate in-game actions in a single player online game. FIG. 3 illustrates an example validation system 220 forming part of the game networking system 150 of FIG. 2. The validation system 220 may comprise a number of hardware-implemented modules provided by one or more processors. The validation system 220 may include a receiving module 304 to receive a validation request 222 (see FIG. 2) that includes: one or more action identifiers to indicate corresponding in-game actions executed on the client system 130; and provisional game state information with respect to the game state of the player 101 and execution of the one or more in-game actions indicated by the validation request 222. In some embodiments, the validation request 222 may include player inputs received at the client system 130 to cause execution of the relevant in-game actions.

[0041] The validation system 220 may further include a validation engine 224 that comprises coded validation logic 228 to execute validation actions identical to the in-game actions identified in the validation request 222. The validation logic 228 is identical to the game logic 212 forming part of the game engine 204. As used herein, the term “identical” with respect to coded logic means not only that identical operations are automatically performed responsive to identical inputs, but also means that the code of the respective coded logic is in the same format and/or computer programming language and may thus be used interchangeably. In an example embodiment, the game logic 212 and the validation logic 228 may be identical sequences of ActionScript code. In some instances, the validation engine 224 may be identical to the game engine 204, while, in other example embodiments, the validation engine 224 and the game engine 204 may be different, but may have identical game logic 212 and validation logic.

[0042] The validation engine 224 may further include a game state library 232 to temporarily hold game state information with respect to the player 101. The validation logic 228 and the game state library 232 may be configured to call operate, so that the validation logic 228 executes validation actions based at least in part on game state information stored in the game state library 232, and updates or changes the game state information in the game state library 232 based on the validation actions.

[0043] A comparison module 308 (FIG. 3) may further form part of the validation system 220 to compare results of execution of the validation actions (e.g., verification game state information resulting from execution of the validation actions) by the validation logic 228 to the provisional game state information included in the validation request 222. The comparison module 308 may be configured to validate the relevant in-game actions when the provisional game state information is identical to the verification game state information, e.g., if performance of the validation actions by the validation engine 224 has the same effect on game state information as execution of the corresponding in-game actions by the game engine 204.

[0044] Referring to FIG. 2, the game networking system 150 may further comprise a game management system 240 comprising a game state database 244 in which validated game states for a plurality of players may be persisted or stored. The game management system 240 further comprises a synchronization manager 248 to receive update requests 252 from client systems 130 to generate and send validation request 222 to the validation system 220, to receive a validation result 256 from the validation system 220, and to persist validated game state information in the game state database 244. In the example embodiment shown in FIG. 2, the synchronization manager 248 is a Web server.

[0045] FIG. 4 illustrates a number of components of an exemplary game management system 240, in this example embodiment being provided by the Web server providing the synchronization manager 248. The game management system 240 includes a receiving module 404 to receive an update request 252 (see FIG. 2) that includes action identifiers indicating a number of in-game actions executed on the game engine 204 of the client system 130, and further includes provisional game state information resulting from performance of the relevant in-game actions. A game state retrieval module 424 may further be included to retrieve prior game state information that indicates previously validated game state information immediately before execution of the relevant in-game actions indicated in the update request 252. The game management system 240 may further include a validation request module 408 to generate and transmit a validation request 222 in response to reception of the update request 252. The game management system 240 further includes a validation result receiver 412 to receive a validation result 256 (FIG. 2) that indicates whether or not in-game actions included in the validation request 222 have been validated. An update module 416 may further be provided to generate validated game state information based at least in part on the provisional game state information included in the update request 252 and based at least in part on the validation result 256, and a persistence module 420 may be provided to
process the updated game state information and to persist the updated game state information to the game state database 244.

[0046] Functionality of the validation system 220, the game management system 240 and their respective components, in accordance with an example embodiment, are further described below with respect to example methods.

Example Methods

[0047] FIG. 5A shows a flowchart 500 of a high-level view of an example method, performed by a validation system such as validation system 220 (FIG. 2), to validate in-game actions performed in a multiplayer online game. The method of flowchart 500 comprises receiving from the game management system 240 action identifiers, at operation 504, and provisional game state information, at operation 508. The action identifiers and provisional game state information may be included in a validation request 222 that may be a serialized communication, for example being in XML format. The action identifiers may be with respect to a plurality of in-game actions performed by the game logic 212, the provisional game state information being with respect to changes to the game state associated with the player 101 caused by execution of the in-game actions. The validation engine 224 may thereafter execute validation actions, at operation 512, by use of the validation logic 228. Results of execution of the validation actions, e.g., in the form of verification game state information generated by the validation engine 224, are compared, at operation 516, to the provisional game state information included in the validation request 222, to validate the in-game actions indicated by the validation request 222. If the provisional game state information with respect to a particular in-game action is identical upon verification game state information for a corresponding validation action, then the particular in-game actions may be validated.

[0048] FIG. 5B shows a flowchart 520 of a high-level view of an example method, performed by a game management system such as game management system 240 (FIG. 2), to validate in-game actions performed. In a multiplayer online game. The method of flowchart 520 comprises receiving from the client system 130 a plurality of action identifiers, at operation 524, and provisional game state information, at operation 528. The action identifiers and provisional game state information may be included in an update request 252 (FIG. 2) that may be a serialized message, in the present example embodiment being in XML format. A validation request 222 may thereafter be transmitted, at operation 532, to the validation system 220, to validate the in-game actions indicated by the action identifiers in the update request 252 by executing validation actions identical to the in-game actions by the validation logic 228 that is identical to the game logic 212 of the game engine 204, on which the in-game actions were initially executed. The game management system 240 may subsequently receive a validation result 256 (FIG. 2) from the validation system 220, at operation 536, indicating that one or more of the in-game actions have been validated. The game management system 240 may thereafter generate updated game state information, at operation 540, based on the validation result 256 and based upon prior game state information that indicates previously validated game state information immediately before execution of the relevant in-game actions. The updated game state information is then persisted, at operation 544.

[0049] FIG. 6 shows a more detailed flowchart 600 of a method to validate in-game actions in a multiplayer online game, and to synchronize game state information in the multiplayer online game. The method of flowchart 600 may be implemented in one embodiment by example system 100 of FIG. 1, with some of the operations being performed by example game management system 240 or example validation system 220 described with reference to FIGS. 3 and 4 above, in the game environment described with reference to FIGS. 1 and 2 above. The flowchart of FIG. 6 is illustrated as being divided into so-called swim lanes, to indicate which of the operations are performed by the client system 130, the game management system 240, or the validation system 220, respectively.

[0050] At operation 604, the player 101 may access the multiplayer online game on the client system 130. Although not illustrated in the flowchart 600 of FIG. 6, the client system 130 may access a webpage hosted by social networking system 140 on the game networking system 150, whereafter the first player’s social networking information may be accessed. In particular embodiments, social networking information on the social networking system 140, the game networking system 150, or both may be accessed. At operation 608, the player 101 may select a game instance to access. In particular embodiments, game instances can be selected from a set of game instances associated with the first player’s friends in the relevant social network. Here, the player 101 selects the game instance uniquely associated with him/her. At operation 612, the game engine 204 may then load the game instance associated with the player 101. Loading of the game instance, at operation 612, may include retrieving a last synchronized or validated game state or game state information from the game management system 240, and loading of the game state information into the game state library 216 of the game engine 204 provided by the client system 130. In the present example embodiment, the game state library 216 is a PHP: Hypertext Preprocessor (PHP) library. Loading of the game instance may further include generating, by use of the game view logic 208 (FIG. 2), a user interface including a game display that includes a visual representation of a virtual in-game environment of the game instance.

[0051] The player 101 may provide gameplay input, at operation 616, to the client system 130, the input, for example, being encountered by the game view logic 208. Upon receipt of input to perform an action that may affect game state, one or more assumption checks or prerequisite checks may be performed, at operation 618, to check whether or not predefined prerequisites or assumptions are satisfied in order to perform the relevant actions. Such prerequisite checks may include, for example, checking whether or not the player character has a required minimum in-game experience level, whether an in-game object on which the action is to be performed has a predefined status to permit performance of the action, and so forth. If the predefined prerequisite check(s) for a particular in-game action is satisfied, the action is executed, at operation 620.

[0052] In-game actions that may affect game state information of the player 101 may be executed, at operation 620, by the game logic 212 of the game engine 204, the game logic 212 in this instance being ActionScript code. The game logic 212 may be configured such that in-game actions that affect game state are modified into atomic logic elements that are referred to herein, inter alia, as “steps.” In FIG. 2, an exemplary sequence of in-game actions or steps are illustrated as...
In some embodiments, logic common to the game engine 204 and the validation engine 224 (e.g., game logic 212 and identical validation logic 228 in the example embodiment of FIG. 2) may be limited to game logic that affects game state, while other game logic (e.g., game logic that does not affect game state information) may located on the client system 130 only.  

Upon execution of each in-game action or step, at operation 620, the game engine 204 may modify the game state information in the game state library 216, and may additionally record the step or action and its resulting game state delta or game state change(s), at operation 624, in an update log 215 (FIG. 2) forming part of the client system 130. In the present example embodiment, the game state deltas or game state changes recorded in the update log 215 may be game step deltas respectively indicating changes to the game state information caused by performance of a corresponding one of the in-game actions or steps.  

In the present example embodiment, game state information in the game state library 216 may be maintained in a property tree or graph with respect to the player 101. Such a property tree may be maintained, for example by the game management system 240, with respect to each of the multiplicity of players of the multiplayer online game, and may be loaded into the game state library 216 upon loading of an associated game instance by the game engine 204. The property tree may contain nested sets of property values that are organized locally. Each property of the property tree is defined by a respective key, which may be a dot-separated list of tree nodes and a final leaf name. Both nodes and names are strings consisting of lowercase letters, numbers, dashes or underscores. An example property tree for a simple slot machine game may look as follows:  

```xml
<properties>
  <player>
    <name type="string">John Doe</name>
    <high_score type="int">3500</high_score>
    <date type="string">2009.11.25 3:22:09</date>
  </player>
  <inventory>
    <coins type="int">274</coins>
  </inventory>
  <achievements.match_one.state:int
</properties>
```

An XML version of the above exemplified property tree, may read as follows:

```xml
<properties user_id="1:12345">... strata state</properties>
<sync>
  <property name="sound">
    <type:boolean type="false"></value>
  </property>
</sync>
<step type="con.zynga.kingdoms.StepsMovement">
  <input>to:5</input>
  <output>
    <property name="locations">
      <type:int type="5"></value>
    </property>
    <property name="energy">
      <type:int type="#"></value>
    </property>
  </output>
</step>
```

Game step information written to the update log 215 may include inputs received for the respective actions or steps, and game step deltas in the form of property changes to the property tree or game state information resulting from execution of the associated action or step. The client system 130 may intermittently or periodically generate checkpoints or update requests 252, at operation 628, with respect to in-game actions or steps which have not yet been validated and/or which have not been included in a previous update request 252, and may transmit the update request 252 to the game management system 240.  

The game client system 130 may produce a serialized update request 252, for example in XML format. Each update request 252 may include action identifiers for the associated in-game actions, as well as proportional game state information in the form of game step deltas resulting from the respective actions. The action identifiers may comprise an action type or step type identifier, as well as identification of user input that prompted performance of the action. Provisional game state information may be provided for each action included in the update request. Such action of a specific game state information may comprise an absolute value for a particular property after completion of the action, or it may comprise a change for Delta to a particular property resulting from performance of the action. The provisional game state information may comprise only those properties of the player character that have been affected by performance of the respective action. An example serialized update request 252 may read as follows:

```xml
<checkpoint>
  <properties user_id="1:12345">... strata state</properties>
  <sync>
    <property name="sound">
      <type:boolean type="false"></value>
    </property>
  </sync>
  <step type="con.zynga.kingdoms.StepsMovement">
    <input>to:5</input>
    <output>
      <property name="locations">
        <type:int type="5"></value>
      </property>
      <property name="energy">
        <type:int type="#"></value>
      </property>
    </output>
  </step>
```

Game step information written to the update log 215 may include inputs received for the respective actions or steps, and game step deltas in the form of property changes to the property tree or game state information resulting from execution of the associated action or step. The client system 130 may intermittently or periodically generate checkpoints or update requests 252, at operation 628, with respect to in-game actions or steps which have not yet been validated and/or which have not been included in a previous update request 252, and may transmit the update request 252 to the game management system 240.
[0064] It will be noted that the above example XML checkpoint or update request 252 includes a synchronization that blindly sets the player state to advance non-validated player state between steps. The example update request 252 above identifies two in-game actions. The first action is identified by an action type identifier as being a movement action. The action identifiers further include an input having a value of 5. Provisional game state information in the form of game state information for the first action comprises location and energy level outputs resulting from performance of the first action. Thus, the player character’s location has a value of 5 after performance of the first movement action, while the energy level of the player character is decremented by a single unit due to the action. The second action indicated by action identifiers in the above example XML update request 252 comprises a number movement action having an input value of 6 and an output value of 6, while the performance of the second movement action results in decrementing of the energy level by a further unit.

[0065] Update requests 252 such as that exemplified above may be generated and transmitted at regular intervals (for example at intervals of 30 seconds to two minutes), or may be generated and transmitted intermittently. In instances of offline play input, information, output information, and game state deltas may be stored in the update log 215 until the game client system 130 is reconnected to the game management system 240, at which time the relevant information with respect to all in-game actions which have not yet been included in a transmitted update request 252 may be included in a single update request 252.

[0066] In some instances, some of the in-game actions indicated in the update request 252 may comprise actions or events having an element of randomness. For example when a player character engages in an in-game battle with a non-player character, the game engine 204 may generate a random seed as an input before execution of the battle. The update request 252 may in such instances include the randomly generated input, or random seed, related to the in-game action or event.

[0067] Upon receipt of the update request 252 by the receiving module 404 (FIG. 4) of the game management system 240, at operation 632, the game state retrieval module 424 (FIG. 4) accesses a persistence subsystem in the form of the game state database 244, and retrieves prior game state information, at operation 634, with respect to the player 101 and the particular game instance indicated by the update request 252. The prior game state information indicates previously validated game state information immediately before execution of the one or more in-game actions indicated in the update request 252. In the present example, the prior game state information includes world state information or environment state information 262 and player state information 266 (FIG. 2). The environment state information 262 may indicate the last validated state of the in-game environment or virtual world, for example indicating the location, type, and states of virtual objects in the in-game environment. The player state information 266 may, for example, be a last validated property tree, such as that described above with respect to the claim client system 130, for the relevant player character. Both of the environment state information 262 and player state information 266 may be in a format which is serializable.

[0068] The method of flowchart 600 may include serializing the prior game state information, at operation 636, and may thereafter include generating and transmitting a validation request 222, at operation 640, by means of the validation request module 408 (FIG. 4). The validation request 222 includes the action identifiers and provisional game state information received in the update request 252 (e.g., action type identifiers, input information, and output information), together with serialized environment state information 262 and player state information 266. The validation request 222 is transmitted, at operation 640, to the validation system 220.

[0069] Upon receipt of the validation request 222, at operation 644, by the receiving module 304 (FIG. 3) of the validation system 220, game state library 232 of the validation engine 224 is populated, at operation 648, with the prior game state information included in the validation request 222 (e.g., environment state information 262 and the player state information 266).

[0070] Thereafter, validation actions corresponding to the in-game actions indicated in the validation request 222 may be performed by the validation logic 228 of the validation engine 224. As mentioned before, the validation logic 228 may comprise ActionScript code identical to that of the game logic 212 forming part of the game engine 204 of the game client system 130, as shown in FIG. 2. Such performance of the validation actions may comprise, with respect to each action, performing a prerequisite check, at operation 652, and, if the prerequisite check is successful, performing the relevant validation action, at operation 656. It will be appreciated that the prerequisite checks may be performed with respect to the prior game state information in the game state library 232. If the prerequisite check fails, the relevant action or step is invalidated, at operation 660.

[0071] Performance of the validation action, at operation 656, may include generation of verification game state information, in this example being validation output resulting from performance of the validation action. Because the validation logic 228 is identical to the game logic 212, execution of identical actions based on identical game state information should result in identical outputs. Game state changes, deltas, or outputs resulting from performance of each validation action may thus be compared, at operation 664, with the corresponding output of the associated in-game action indicated in the validation request 222. If the validation output (also referred to as the results of execution of the relevant validation action) is identical to the corresponding in-game output (also referred to as the provisional game state information), then the in-game action is validated, at operation 668. If, however, there is a discrepancy between the validation output and the corresponding in-game output, then the relevant action is invalidated, at operation 660.
When a particular action is validated, at operation 668, the sequence of operations to validate an action, at operations 652 to 668, is performed for the next in-game action indicated in the validation request 222. The plurality of in-game actions indicated in the validation request 222 are thus validated stepwise, in sequence, until all of the actions have been validated, or until a first invalid action is identified, either by failure of a prerequisite check, at operation 652, or by identification of a discrepancy between the validation output and the in-game output, at operation 664.

When validation of the sequence of in-game actions of the validation request 222 is completed (or when one of the in-game actions are invalidated), the validation result 256 is generated and transmitted, at operation 672, to the game management system 240. Generation of the validation result 256 may comprise assembling or collating a resultant or cumulative game state delta resulting from all of the validated actions or steps. Property values in the validation result 256 may thus indicate a final output (e.g., for a property such as location) or a cumulative delta (e.g., for property such as player character energy or experience) at the last validated action. The validation result 256 may again be serialized, in the example embodiment being an XML document. An extract of an example validation result 256, following validation of all of the actions indicated in the exemplary validation request 222 provided above, may look as follows:

```
<checkpoint success="true">
  ...
</checkpoint>
```

After receiving the validation result 256 at operation 676, the game management system 240 combines the game state deltas for validated game state information indicated in the validation result 256 with the prior game state information, at operation 680, to produce updated game state information that is whole, consistent, new game state information, in the present example embodiment comprising validated environment state information 262 and validated player state information 266. The validated game state information may be stored by being persisted to the game state database 244, at operation 684.

In the event of invalidation of any in-game action by the validation system 220 at operation 660, the game management system 240 may communicate a non-synchronization notification to the client system 130, at operation 688, to notify the client system 130 that the game state information of the client system 130 is out of synchronization with the game management system 240, e.g., that the provisional game state information included in the update request 252 could not be completely validated. The non-synchronization notification may identify the last or furthest validated action and/or may indicate furthest validated for validated state information. In response to receiving the non-synchronization notification, the client system 130 may cause operation of the game engine 204 to stop gameplay, forcing the player to reload the game instance, and returning to a furthest consistent or synchronized state, at operation 692, as indicated in the non-synchronization notification.

The provision of identical game logic 212 and the validation logic 228 on the client-side and server-side, respectively, eliminates the variety of separate validation code on the server side. Using coded game logic on the server side for validating in-game actions may therefore significantly reduce development time for game features that involve changes to the game states. An overall reduction in code required for making available the in-game actions further results in reduced bug counts due in part to smaller code paths, and due to the fact that there is no potential for server/client code mismatch.

The example embodiment described above further promotes the asynchronous operation of the game engine 204 on the client system 130, as the update requests 252 may be sent to the game management system 240 for validation of the in-game actions at any desired level of infrequency. The architecture illustrated in FIG. 2, for example, increases architectural scalability of the system.

**Data Flow**

FIG. 7 illustrates an example data flow between the components of an example system 700. In particular embodiments, system 700 can include client system 730, social networking system 720a, and game networking system 720b. A system 300 such as that described with reference to FIG. 3 may be provided by the client system 730, the social networking system 720a, or the game networking system 720b, or by any combination of these systems. The components of system 700 can be connected to each other in any suitable configuration, using any suitable type of connection. The components may be connected directly or over any suitable network. Client system 730, social networking system 720a, and game networking system 720b can have one or more corresponding data stores such as local data store 725, social data store 745, and game data store 765, respectively. Social networking system 720a and game networking system 720b can also have one or more servers that can communicate with client system 730 over an appropriate network. Social networking system 720a and game networking system 720b can have, for example, one or more Internet servers for communicating with client system 730 via the Internet. Similarly, social networking system 720a and game networking system 720b can have one or more mobile servers for communicating with client system 730 via a mobile network (e.g., GSM, PCS, Wi-Fi, WPAN, etc.), in some embodiments, one server may be able to communicate with client system 730 over both the Internet and a mobile network. In other embodiments, separate servers can be used.

Client system 730 can receive and transmit data 723 to and from game networking system 720b. This data can include, for example, webpages, messages, game inputs,
game displays, HTTP packets, data requests, transaction information, updates, and other suitable data. As discussed with reference to the example embodiments of FIGS. 2-6, selected communications may be serialized documents, such as for example XML documents. At some other time, or at the same time, game networking system 720b can communicate data 743, 747 (e.g., game state information, game system account information, page info, messages, data requests, updates, etc.) with other networking systems, such as social networking system 720a (e.g., Facebook, MySpace, etc.). Client system 730 can also receive and transmit data 727 to and from social networking system 720a. This data can include, for example, webpages, messages, social graph information, social network displays, HTTP packets, data requests, transaction information, updates, and other suitable data.

Communication between client system 730, social networking system 720a, and game networking system 720b can occur in any appropriate electronic communication medium or network using any suitable communications protocols. For example, client system 730, as well as various servers of the systems described herein, may include Transport Control Protocol/Internet Protocol (TCP/IP) networking stacks to provide for datagram and transport functions. Of course, any other suitable network and transport layer protocols can be utilized.

In addition, hosts or end-systems described herein may use a variety of lower layer communications protocols, including client-server (or request-response) protocols, such as HTTP, other communications protocols, such as HTTPS, FTP, SNMP, TELNET, and a number of other protocols may be used. In addition, a server in one interaction context may be a client in another interaction context. In particular embodiments, the information transmitted between hosts may be formatted as HyperText Markup Language (HTML) documents. Other structured document languages or formats can be used, such as XML and the like. Executable code objects, such as JavaScript and ActionScript, can also be embedded in the structured documents.

In some client-server protocols, such as the use of HTML over HTTP, a server generally transmits a response to a request or from a client. The response may comprise one or more data objects. For example, the response may comprise a first data object, followed by subsequently transmitted data objects. In particular embodiments, a client request may cause a server to respond with a first data object, such as an HTML page, which itself refers to other data objects. A client application, such as a browser, will request these additional data objects as it parses or otherwise processes the first data object.

In particular embodiments, an instance of an online game can be stored as a set of game state parameters that characterize the state of various in-game objects, such as, for example, player character state parameters, non-player character parameters, and virtual item parameters. In particular embodiments, game state is maintained in a database as a serialized, unstructured string of text data as a so-called Binary Large Object (BLOB). When a player accesses an online game on game networking system 720b, the BLOB containing the game state for the instance corresponding to the player can be transmitted to client system 730 for use by a client-side executed object to process. In particular embodiments, the client-side executable may be a FLASH-based game, which can de-serialize the game state data in the BLOB. As a player plays the game, the game logic implemented at client system 730 maintains and modifies the various game state parameters locally. The client-side game logic may also batch game events, such as mouse clicks, and transmit these events to game networking system 720b. Game networking system 720b may itself operate by retrieving a copy of the BLOB from a database or an intermediate memory cache (memcache) layer. Game networking system 720b can also de-serialize the BLOB to resolve the game state parameters and execute its own game logic based on the events in the batch file of events transmitted by the client to synchronize the game state on the server side. Game networking system 720b may then re-serialize the game state, now modified, into a BLOB, and pass this to a memory cache layer for lazy updates to a persistent database.

With a client-server environment in which the online games may run, one server system, such as game networking system 720b, may support multiple client systems 730. At any given time, there may be multiple players at multiple client systems 730 playing the same online game. In practice, the number of players playing the same game at the same time may be very large. As the game progresses with each player, multiple players may provide different inputs to the online game at their respective client systems 730, and multiple client systems 730 may transmit multiple player inputs and/or game events to game networking system 720b for further processing. In addition, multiple client systems 730 may transmit other types of application data to game networking system 720b.

In particular embodiments, a computer-implemented game may be a text-based or turn-based game implemented as a series of web pages that are generated after a player selects one or more actions to perform. The web pages may be displayed in a browser client executed on client system 730. As an example and not by way of limitation, a client application downloaded to client system 730 may operate to serve a set of webpages to a player. As another example and not by way of limitation, a computer-implemented game may be an animated or rendered game executable as a stand-alone application or within the context of a web page or other structured document. In particular embodiments, the computer-implemented game may be implemented using Adobe FLASH-based technologies. As an example and not by way of limitation, a game may be fully or partially implemented as a SWF object that is embedded in a web page and executable by a Flash media player plug-in. In particular embodiments, one or more described webpages may be associated with or accessed by social networking system 720a. This disclosure contemplates using any suitable application for the retrieval and rendering of structured documents hosted by any suitable network-addressable resource or website.

Application event data of a game is any data relevant to the game (e.g., player inputs). In particular embodiments, each application datum may have a name and a value, and the value of the application datum may change (i.e., be updated) at any time. When an update to an application datum occurs at client system 730, either caused by an action of a game player or by the game logic itself, client system 730 may need to inform game networking system 720b of the update. For example, if the game is a farming game with a harvest mechanic (such as Zynga Farmville), an event can correspond to a player clicking on a parcel of land to harvest a crop. In such an instance, the application event data may identify an event or action (e.g., harvest) and an object in the game to
which the event or action applies. For illustration purposes and not by way of limitation, system 700 is discussed in reference to updating a multi-player online game hosted on a network-addressable system (such as, for example, social networking system 720a or game networking system 720b), where an instance of the online game is executed remotely on a client system 730, which then transmits application event data to the hosting system such that the remote game server synchronizes the game state associated with the instance executed by the client system 730.

[0087] In a particular embodiment, one or more objects of a game may be represented as an Adobe Flash object. Flash may manipulate vector and raster graphics, and supports bidirectional streaming of audio and video. “Flash” may mean the authoring environment, the player, or the application files. In particular embodiments, client system 730 may include a Flash client. The Flash client may be configured to receive and run Flash application or game object code from any suitable networking system (such as, for example, social networking system 720a or game networking system 720b). In particular embodiments, the Flash client may be run in a browser client executed on client system 730. A player can interact with Flash objects using client system 730 and the Flash client. The Flash objects can represent a variety of in-game objects. Thus, the player may perform various in-game actions on various in-game objects by making various changes and updates to the associated Flash objects. In particular embodiments, in-game actions can be initiated by clicking or similarly interacting with a Flash object that represents a particular in-game object. For example, a player can interact with a Flash object to use, move, rotate, delete, attack, shoot, or harvest an in-game object. This disclosure contemplates performing any suitable in-game action by interacting with any suitable Flash object. In particular embodiments, when the player makes a change to a Flash object representing an in-game object, the client-executed game logic may update one or more game state parameters associated with the in-game object. To ensure synchronization between the Flash object shown to the player at client system 730, the Flash client may send the events that caused the game state changes to the in-game object to game networking system 720b. However, to expedite the processing and hence the speed of the overall gaming experience, the Flash client may collect a batch of some number of events or updates into a batch file. The number of events or updates may be determined by the Flash client dynamically or determined by game networking system 720b based on server loads or other factors. For example, client system 730 may send a batch file to game networking system 720b whenever 50 updates have been collected or after a threshold period of time, such as every minute.

[0088] As used herein, the term “application event data” may refer to any data relevant to a computer-implemented game application that may affect one or more game state parameters, including, for example and without limitation, changes to player data or metadata, changes to player social connections or contacts, player inputs to the game, and events generated by the game logic. In particular embodiments, each application datum may have a name and a value. The value of an application datum may change at any time in response to the game play of a player or in response to the game engine (e.g., based on the game logic). In particular embodiments, an application data update occurs when the value of a specific application datum is changed. In particular embodiments, each application event datum may include an action or event name and a value (such as an object identifier). Thus, each application datum may be represented as a name-value pair in the batch file. The batch file may include a collection of name-value pairs representing the application data that have been updated at client system 730. In particular embodiments, the batch file may be a text file and the name-value pairs may be in string format.

[0089] In particular embodiments, when a player plays an online game on client system 730, game networking system 720b may serialize all the game-related data, including, for example and without limitation, game states, game events, and user inputs, for this particular user and this particular game into a BLOB and store the BLOB in a database. The BLOB may be associated with an identifier that indicates that the BLOB contains the serialized game-related data for a particular player and a particular online game. In particular embodiments, while a player is not playing the online game, the corresponding BLOB may be stored in the database. This enables a player to stop playing the game at any time without losing the current state of the game that the player is in. When a player resumes playing the game next time, game networking system 720b may retrieve the corresponding BLOB from the database to determine the most-recent values of the game-related data. In particular embodiments, while a player is playing the online game, game networking system 720b may also load the corresponding BLOB into a memory cache so that the game system may have faster access to the BLOB and the game-related data contained therein.

Systems and Methods

[0090] In particular embodiments, one or more described webpages may be associated with a networking system or networking service. However, alternate embodiments may have application to the retrieval and rendering of structured documents hosted by any type of network-addressable resource or web site. Additionally, as used herein, a user may be an individual, a group, or an entity (such as a business or third-party application).

[0091] Particular embodiments may operate in a WAN environment, such as the Internet, including multiple network-addressable systems. FIG. 8 illustrates an example network environment 800, in which various example embodiments may operate. Network cloud 860 generally represents one or more interconnected networks, over which the systems and hosts described herein can communicate. Network cloud 860 may include packet-based WAN (such as the Internet), private networks, wireless networks, satellite networks, cellular networks, paging networks, and the like. As FIG. 8 illustrates, particular embodiments may operate in a network environment 800 comprising one or more networking systems, such as social networking system 820a, game networking system 820b, and one or more client systems 830. The components of social networking system 820a and game networking system 820b operate analogously; as such, hereinafter they may be referred to simply as networking system 820. Client systems 830 are operably connected to the network environment 800 via a network service provider, a wireless carrier, or any other suitable means.

[0092] Networking system 820 is a network-addressable system that, in various example embodiments, comprises one or more physical servers 822 and data stores 824. The one or more physical servers 822 are operably connected to network cloud 860 via, by way of example, a set of routers and/or
networking switches 826. In an example embodiment, the functionality hosted by the one or more physical servers 822 may include web or HTTP servers, FTP servers, and, without limitation, webpages and applications implemented using Common Gateway Interface (CGI) script, PHP Hyper-text Preprocessor (PHP), Active Server Pages (ASP), HTML, XML, Java, JavaScript, Asynchronous JavaScript and XML (AJAX), Flash, ActionScript, and the like.

Physical servers 822 may host functionality directed to the operations of networking system 820. Hereinafter servers 822 may be referred to as server 822, although server 822 may include numerous servers hosting, for example, networking system 820, as well as other content distribution servers, data stores, and databases. Data store 824 may store content and data relating to, and enabling, operation of networking system 820 as digital data objects. A data object, in particular embodiments, is an item of digital information typically stored or embodied in a data file, database, or record. Content objects may take many forms, including text (e.g., ASCII, SGML, HTML), images (e.g., jpeg, tif and gif), graphics (vector-based or bitmap), audio, video (e.g., mpeg), or other multimedia, and combinations thereof. Content object data may also include executable code objects (e.g., games executable within a browser window or frame), podcasts, and the like. Logically, data store 824 corresponds to one or more sets of integrated databases, that maintain information as an integrated collection of logically related records or files stored on one or more physical systems. Structurally, data store 824 generally includes one or more of a large class of data storage and management systems in particular embodiments. Data store 824 may be implemented by any suitable physical system(s) including components, such as one or more database servers, mass storage media, media library systems, storage area networks, data storage clouds, and the like. In one example embodiment, data store 824 includes one or more servers, databases (e.g., MySQL), and/or data warehouses. Data store 824 may include data associated with different networking system 820 users and/or client systems 830.

[0094] Client system 830 is generally a computer or computing device including functionality for communicating (e.g., remotely) over a computer network. Client system 830 may be a desktop computer, laptop computer, personal digital assistant (PDA), in- or out-of-car navigation system, smart phone or other cellular or mobile phone, or mobile gaming device, among other suitable computing devices. Client system 830 may execute one or more client applications, such as a web browser (e.g., Microsoft Internet Explorer, Mozilla Firefox, Apple Safari, Google Chrome, and Opera), to access and view content over a computer network. In particular embodiments, the client applications allow a user of client system 830 to enter addresses of specific network resources to be retrieved, such as resources hosted by networking system 820. These addresses can be Uniform Resource Locators (URL's) and the like. In addition, once a page or other resource has been retrieved, the client applications may provide access to other pages or records when the user “clicks” on hyperlinks to other resources. By way of example, such hyperlinks may be located within the webpages and provide an automated way for the user to enter the URL of another page and to retrieve that page.

[0095] A webpage or resource embedded within a webpage, which may itself include multiple embedded resources, may include data records, such as plain textual information, or more complex digitally encoded multimedia content, such as software programs or other code objects, graphics, images, audio signals, videos, and the like. Other prevalent markup language for creating webpages is HTML. Other common web browser-supported languages and technologies include XML, the Extensible Hyper-text Markup Language (XHTML), JavaScript, Flash, ActionScript, Cascading Style Sheet (CSS), and, frequently, Java. By way of example, HTML enables a page developer to create a structured document by denoting structural semantics for text and links, as well as images, web applications, and other objects that can be embedded within the page. Generally, a webpage may be delivered to a client as a static document; however, through the use of web elements embedded in the page, an interactive experience may be achieved with the page or a sequence of pages. During a user session at the client, the web browser interprets and displays the pages and associated resources received or retrieved from the website hosting the page, as well as, potentially, resources from other websites.

[0096] When a user at a client system 830 desires to view a particular webpage (hereinafter referred to as target structured document) hosted by networking system 820, the user’s web browser, or other document rendering engine or suitable client application, formulates and transmits a request to networking system 820. The request generally includes a URL or other document identifier as well as metadata or other information. By way of example, the request may include information identifying the user, such as a user ID, as well as information identifying or characterizing the web browser or operating system running on the user’s client computing device 830. The request may also include location information identifying a geographic location of the user’s client system or a logical network location of the user’s client system. The request may also include a timestamp identifying when the request was transmitted.

[0097] Although the example network environment 800 described above and illustrated in FIG. 8 is described with respect to social networking system 820a and social networking system 820b, this disclosure encompasses any suitable network environment using any suitable systems. As an example and not by way of limitation, the network environment may include online media systems, online reviewing systems, online search engines, online advertising systems, or any combination of two or more such systems.

[0098] FIG. 9 illustrates an example computing system architecture, which may be used to implement a server 822 or a client system 830. In one embodiment, hardware system 900 comprises a processor 902, a cache memory 904, and one or more executable modules and drivers, stored on a tangible computer-readable medium, directed to the functions described herein. Additionally, hardware system 900 may include a high performance input/output (I/O) bus 906 and a standard I/O bus 908. A host bridge 910 may couple processor 902 to high performance I/O bus 906, whereas I/O bus bridge 912 couples the two buses 906 and 908 to each other. A system memory 914 and one or more network/communication interfaces 916 may couple to bus 906. Hardware system 900 may further include video memory (not shown) and a display device coupled to the video memory. Mass storage 918 and I/O ports 920 may couple to bus 908. Hardware system 900 may optionally include a keyboard, a pointing device, and a display device (not shown) coupled to bus 908. Collectively, these elements are intended to represent a broad...
category of computer hardware systems, including but not limited to general purpose computer systems based on the x86-compatible processors manufactured by Intel Corporation of Santa Clara, Calif., and the x86-compatible processors manufactured by Advanced Micro Devices (AMD), Inc., of Sunnyvale, Calif., as well as any other suitable processor.

The elements of hardware system 900 are described in greater detail below. In particular, network interface 916 provides communication between hardware system 900 and any of a wide range of networks, such as an Ethernet (e.g., IEEE 802.3) network, a backplane, and the like. Mass storage 918 provides permanent storage for the data and programming instructions to perform the above-described functions implemented in servers 822, whereas system memory 914 (e.g., DRAM) provides temporary storage for the data and programming instructions when executed by processor 902. I/O ports 920 are one or more serial and/or parallel communication ports that provide communication between additional peripheral devices, which may be coupled to hardware system 900.

Hardware system 900 may include a variety of system architectures, and various components of hardware system 900 may be rearranged. For example, cache 904 may be on-chip with processor 902. Alternatively, cache 904 and processor 902 may be packed together as a "processor module," with processor 902 being referred to as the "processor core." Furthermore, certain embodiments of the present disclosure may not require nor include all of the above components. For example, the peripheral devices shown coupled to standard I/O bus 908 may couple to high-performance I/O bus 906. In addition, in some embodiments, only a single bus may exist, with the components of hardware system 900 being coupled to the single bus. Furthermore, hardware system 900 may include additional components, such as additional processors, storage devices, or memories.

An operating system manages and controls the operation of hardware system 900, including the input and output of data to and from software applications (not shown). The operating system provides an interface between the software applications being executed on the system and the hardware components of the system. Any suitable operating system may be used, such as the LINUX Operating System, the Apple Macintosh Operating System, available from Apple Computer Inc. of Cupertino, Calif., UNIX operating systems, Microsoft(r) Windows(r) operating systems, BSD operating systems, and the like. Of course, other embodiments are possible. For example, the functions described herein may be implemented in firmware or on an application-specific integrated circuit (ASIC).

Furthermore, the above-described elements and operations can be comprised of instructions that are stored on non-transitory storage media. The instructions can be retrieved and executed by a processing system. Some examples of instructions are software, program code, and firmware. Some examples of non-transitory storage media are memory devices, tape, disks, integrated circuits, and servers. The instructions are operational when executed by the processing system to direct the processing system to operate in accord with the disclosure. The term "processing system" refers to a single processing device or a group of inter- operational processing devices. Some examples of processing devices are integrated circuits and logic circuitry. Those skilled in the art are familiar with instructions, computers, and storage media.

Modules, Components, and Logic

Certain embodiments are described herein as including logic or a number of components, modules, or mechanisms. Modules may constitute either software modules (e.g., code embodied (1) on a non-transitory machine-readable medium or (2) in a transmission signal) or hardware-implemented modules. A hardware-implemented module is a tangible unit capable of performing certain operations and may be configured or arranged in a certain manner. In example embodiments, one or more computer systems (e.g., a standalone, client or server computer system) or one or more processors may be configured by software (e.g., an application or application portion) as a hardware-implemented module that operates to perform certain operations as described herein.

In various embodiments, a hardware-implemented module may be implemented mechanically or electronically. For example, a hardware-implemented module may comprise dedicated circuitry or logic that is permanently configured (e.g., as a special-purpose processor, such as a field programmable gate array (FPGA) or an ASIC) to perform certain operations. A hardware-implemented module may also comprise programmable logic or circuitry (e.g., as encompassed within a general-purpose processor or other programmable processor) that is temporarily configured by software to perform certain operations. It will be appreciated that the decision to implement a hardware-implemented module mechanically, in dedicated and permanently configured circuitry, or in temporarily configured circuitry (e.g., configured by software) may be driven by cost and time considerations. Accordingly, the term "hardware-implemented module" should be understood to encompass a tangible entity, be that an entity that is physically constructed, permanently configured (e.g., wired) or temporarily or transitorily configured (e.g., programmed) to operate in a certain manner and/or to perform certain operations described herein. Considering embodiments in which hardware-implemented modules are temporarily configured (e.g., programmed), each of the hardware-implemented modules need not be configured or instantiated at any one instance in time. For example, where the hardware-implemented modules comprise a general-purpose processor configured using software, the general-purpose processor may be configured as respective different hardware-implemented modules at different times. Software may accordingly configure a processor, for example, to constitute a particular hardware-implemented module at one instance of time and to constitute a different hardware-implemented module at a different instance of time.

Hardware-implemented modules can provide information to, and receive information from, other hardware-implemented modules. Accordingly, the described hardware-implemented modules may be regarded as being communicatively coupled. Where multiple of such hardware-implemented modules exist contemporaneously, communications may be achieved through signal transmission over appropriate circuits and buses that connect the hardware-implemented modules. In embodiments in which multiple hardware-implemented modules are configured or instantiated at different times, communications between such hardware-implemented modules may be achieved, for example, through the storage and retrieval of information in memory structures to which the multiple hardware-implemented modules have access. For example, one hardware-implemented
module may perform an operation, and store the output of that operation in a memory device to which it is communicatively coupled. A further hardware-implemented module may then, at a later time, access the memory device to retrieve and process the stored output. Hardware-implemented modules may also initiate communications with input or output devices, and can operate on a resource (e.g., a collection of information).

The various operations of example methods described herein may be performed, at least partially, by one or more processors that are temporarily configured (e.g., by software) or permanently configured to perform the relevant operations. Whether temporarily or permanently configured, such processors may constitute processor-implemented modules that operate to perform one or more operations or functions. The modules referred to herein may, in some example embodiments, comprise processor-implemented modules.

Similarly, the methods described herein may be at least partially processor-implemented. For example, at least some of the operations of a method may be performed by one or more processors or processor-implemented modules. The performance of certain of the operations may be distributed among the one or more processors, not only residing within a single machine, but deployed across a number of machines. In some example embodiments, the processor or processors may be located in a single location (e.g., within a home environment, an office environment or as a server farm), while in other embodiments the processors may be distributed across a number of locations.

The one or more processors may also operate to support performance of the relevant operations in a “cloud computing” environment or as a “software as a service” (SaaS). For example, at least some of the operations may be performed by a group of computers (as examples of machines including processors), with these operations being accessible via a network (e.g., the Internet) and via one or more appropriate interfaces (e.g., Application Program Interfaces (APIs)).

Miscellaneous

One or more features from any embodiment may be combined with one or more features of any other embodiment without departing from the scope of the disclosure.

A recitation of “a,” “an,” or “the” is intended to mean “one or more” unless specifically indicated to the contrary.

The present disclosure encompasses all changes, substitutions, variations, alterations, and modifications to the example embodiments herein that a person having ordinary skill in the art would comprehend. Similarly, where appropriate, the appended claims encompass all changes, substitutions, variations, alterations, and modifications to the example embodiments herein that a person having ordinary skill in the art would comprehend.

For example, the methods described herein may be implemented using hardware components, software components, and/or any combination thereof. By way of example, while embodiments of the present disclosure have been described as operating in connection with a networking website, various embodiments of the present disclosure can be used in connection with any communications facility that supports web applications. Furthermore, in some embodiments the term “web service” and “website” may be used interchangeably and additionally may refer to a custom or generalized API on a device, such as a mobile device (e.g., a cellular phone, smartphone, personal GPS, personal digital assistant, personal gaming device), that makes API calls directly to a server. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. It will, however, be evident that various modifications and changes may be made thereunto without departing from the broader spirit and scope of the disclosure as set forth in the claims and that the disclosure is intended to cover all modifications and equivalents within the scope of the following claims.

1. A computer-implemented method to validate in-game actions performed in a computer-implemented game, the method comprising:
executing at a server-side device one or more validation actions corresponding to one or more in-game actions performed at a client-side device in an instance of the computer-implemented game, the one or more validation actions being executed using validation code identical to game code used to execute the one or more in-game actions; and
invalidating at least one of the one or more in-game actions responsive to malperformance of at least one corresponding validation action.

2. (Canceled)

3. The computer-implemented method of claim 1, wherein execution of at least some of the one or more validation actions includes performing a prerequisite check to determine whether one or more predefined game state requirements for the corresponding in-game action are satisfied, malperformance of a particular one of the one or more validation actions comprising failure of a corresponding prerequisite check for the particular validation action.

4. The computer-implemented method of claim 3, wherein the one or more validation actions comprises a sequence of validation actions corresponding to a sequence of in-game actions, failure of the prerequisite check for the particular validation action automatically resulting in nonperformance of validation actions following the particular validation action in the sequence of validation actions.

5. The computer-implemented method of claim 1, wherein execution of each of the one or more validation actions produces validation game state information, the method further comprising:
comparing, for each executed validation action, the validation game state information with corresponding provisional game state information associated with the corresponding in-game action; and
validating each one of the one or more in-game actions for which the provisional game state information and the corresponding validation game state information are identical.

6. The computer-implemented method of claim 5, wherein malperformance of the at least one validation action comprises determining that the provisional game state information and the validation game state information corresponding to the at least one validation action are not identical.

7. The computer-implemented method of claim 5, wherein the provisional game state information and the validation game state information comprises a game state delta that indicates one or more changes to a game state caused by performance of the associated in-game action and the associated validation action, respectively.
8. The computer-implemented method of claim 7, further comprising receiving a validation request that includes:
one or more action identifiers respectively indicating the one or more in-game actions to be validated; and
provisional game state information for each of the one or more in-game actions.

9. The computer-implemented method of claim 8, wherein the validation request further includes prior game state information that indicates game state information corresponding to a most recent validated in-game action prior to performance of any of the one or more in-game actions indicated in the validation request.

10. The computer-implemented method of claim 9, further comprising populating a game state library of a validation engine that includes the validation code with the prior game state information, execution of the one or more validation actions being performed by using the validation code in cooperation with the game state library.

11. A system to validate in-game actions performed in a computer-implemented game, the system comprising a server-side validation engine to:
execute one or more validation actions corresponding to one or more in-game actions performed at a client-side device in an instance of the computer-implemented game, the one or more validation actions being executed using validation code identical to game code used to execute the one or more in-game actions; and
invalidate at least one of the one or more in-game actions responsive to malperformance of at least one corresponding validation action.

12. (canceled)

13. The system of claim 11, wherein the validation engine is configured such that execution of at least some of the one or more validation actions includes performing a prerequisite check to determine whether one or more predefined game state requirements for the corresponding in-game action are satisfied, malperformance of a particular one of the one or more validation actions comprising failure of a corresponding prerequisite check for the particular validation action.

14. The system of claim 13, wherein the one or more validation actions comprises a sequence of validation actions corresponding to a sequence of in-game actions, the validation engine being configured such that failure of the prerequisite check for the particular validation action automatically results in nonperformance of validation actions following the particular validation action in the sequence of validation actions.

15. The system of claim 11, wherein the validation engine is configured to produce validation game state information resulting from execution of each of the one or more validation actions, the system further comprising a comparison module to compare, for each executed validation action, the validation game state information with corresponding provisional game state information associated with the corresponding in-game action, the validation engine further being configured to validate each one of the one or more in-game actions for which the provisional game state information and the corresponding validation game state information are identical.

16. The system of claim 15, wherein malperformance of the at least one validation action comprises determination by the comparison module that the provisional game state information and the validation game state information corresponding to the at least one validation action are not identical.

17. The system of claim 15, wherein the provisional game state information and the validation game state information comprises a game state delta that indicates one or more changes to a game state caused by performance of the associated in-game action and the associated validation action, respectively.

18. The system of claim 17, further comprising a receiving module to receive a validation request that includes:
one or more action identifiers respectively indicating the one or more in-game actions to be validated; and
provisional game state formation for each of the one or more in-game actions.

19. The system of claim 18, wherein the validation request further includes prior game state information that indicates game state information corresponding to a most recent validated in-game action prior to performance of any of the one or more in-game actions indicated in the validation request.

20. A non-transitory machine-readable storage medium storing instructions which, when performed by a machine, cause the machine to:
execute at a server-side device one or more validation actions corresponding to one or more in-game actions performed at a client-side device in an instance of the computer-implemented game, the one or more validation actions being executed using validation code identical to game code used to execute the one or more in-game actions; and
invalidate at least one of the one or more in-game actions responsive to malperformance of at least one corresponding validation action.