A **Timeline** object is used to maintain an ordered list of **TimelineEvent**:

|  |  |
| --- | --- |
| **Event Type** | **Description** |
| etype\_None | Default event. Ignored by the controller |
| etype\_Announce | The start of an attack. Notify all the Pokemon’s opponents. |
| etype\_Free | The Pokemon is free for a new **Action**. |
| etype\_Fast | The activation of the Pokemon’s Fast Attack. Deals damage to all its opponents. |
| etype\_Charged | The activation of the Pokemon’s Charged attack. Deals damage to all its opponents. |
| etype\_Dodge | The Pokemon performs dodging. |
| etype\_Enter | The Pokemon enters the battle. |

An **Action** object is used to communicate between the controller and the strategy functions. The controller asks each Player what to do every now and then. The Player should response with an Action object. The controller will then execute the action (or buffer it), which is essentially queueing appropriate **TimelineEvent** objects accordingly. The available types of Action includes:

|  |  |
| --- | --- |
| **Action Type** | **Description** |
| atype\_None | Default action. Ignored by the controller |
| atype\_Wait | Wait and do nothing. Can wait either finitely (set Action.value > 0) or infinitely (set Action.value <= 0). |
| atype\_Fast | Use Fast attack. |
| atype\_Charged | User Charged Attack. Action.value is used to index which Charged Move to use. Set Action.value < 0 to use the last Charged Move. |
| atype\_Dodge | Perform a dodge. |
| atype\_Switch | Switch the current Pokemon to another Pokemon in the current party indexed by Action.value. |

Initially,

* For each player:
  + Set current\_action to None
  + Set buffer\_action to None
  + Put Enter event for head Pokemon

**Register\_and\_execute (player, action)**:

* Set current\_action to action
* Set current\_action.time to current time
* Call corresponding action register method

**handle\_event\_free**():

* If the player’s buffer\_action is None:
  + Call on\_free() to get response action
  + Register\_and\_execute that response action
* Else:
  + Register\_and\_execute buffer\_action
  + Set buffer\_action to None
* If on\_clear is not NULL:
  + Call on\_clear() to get response action
  + Set buffer\_action to that response action

**handle\_event\_announce**():

* For each rival player with on\_attack not NULL:
  + Call on\_attack() to get response action
  + If current\_action is None or Wait:
    - Register\_and\_execute the response action
  + Else:
    - Set buffer\_action to the response action