
Programming Games Contests Framework

Release 0.0.1

Oct 13, 2022

Contents:

1	Programming Games Contests Framework	3
1.1	Installation	3
1.2	Usages	3
1.3	Contribution	3
2	Quick start	5
3	progames	7
3.1	progames package	7
4	Indices and tables	17
	Python Module Index	19
	Index	21

You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

Programming Games Contests Framework

1.1 Installation

clone from git hub

```
git clone git@github.com:ferez96/Progames.git
```

install

```
cd Progames/  
pip install .
```

1.2 Usages

init db

```
flask --app progames.server init-db
```

start http server

```
flask --app progames.server run
```

1.3 Contribution

Feel free to create Issues & Pull Requests

Or contact me via: [Facebook](#)

CHAPTER 2

Quick start

3.1 progames package

3.1.1 Subpackages

progames.core package

Subpackages

progames.core.engine package

Submodules

progames.core.engine.abc module

```
class progames.core.engine.abc.Store
    Bases: collections.abc.Sized
    empty () → bool
        check if the store is empty
    pop (*args, **kwargs) → Item
        get the first item

        Returns the first item
    store (entity: Item, *args, **kwargs) → None
        store item

        Parameters entity – item to store
class progames.core.engine.abc.GameWorld
    Bases: object
```

accept (*command*: *progames.core.engine.messages.Command*) → *Iter-*
able[*progames.core.engine.messages.Event*]

Parameters **command** – command

rollback (*command*: *progames.core.engine.messages.Command*) → *None*

Parameters **command** – the failed command which causes rollback

update () → *None*

Some game need update game world each frame

class *progames.core.engine.abc.Message*

Bases: *object*

progames.core.engine.concurrent module

progames.core.engine.concurrent.run (*func*, *args=None*, *kwargs=None*)
execute a function in default thread pool

Parameters

- **func** (*function*) – function to be executed
- **args** (*tuple*) – args
- **kwargs** (*dict*) – kwargs

Returns *None*

progames.core.engine.messages module

class *progames.core.engine.messages.Command*

Bases: *progames.core.engine.abc.Message*

base command

visit (*world*: *progames.core.engine.abc.GameWorld*) → *Iterable*[*progames.core.engine.messages.Event*]

class *progames.core.engine.messages.Event*

Bases: *progames.core.engine.abc.Message*

base event

class *progames.core.engine.messages.GameEnded*

Bases: *progames.core.engine.messages.Event*

default event raise when game is ended

progames.core.engine.processors module

class *progames.core.engine.processors.Processor* (*game*: *progames.core.engine.GameWorld*,
store: *progames.core.engine.Store*,
args*, *kwargs*)

Bases: *object*

Process game and produce events

Notes

- Player's input data will be processed and transform to commands.
- Each frame, game only consume 1 command from command store.
- Command is a visitor, travel around game world and produce events. These events will be collected and sent to listeners.
- Rollback is required to implement

command_store = None
command store

cps = 50
commands per seconds (max), $0 < \text{cps}$

get_next_command() → typing.Optional[progames.core.engine.Command]
get command to process
:raise RuntimeError

handle (*event: progames.core.engine.Event*) → None
base handle event, don't forget call: *super().handle(event)*

is_over() → bool
check if game is over

over = None
flag for checking if game is over

process_command() → None
process commands one by one

Flows:

1. pop 1 command from command store
2. the command go through game world, make changes and produce events
3. notify events to listeners (register)
4. update game world

Raises

- **RuntimeError** – game must stop
- **RuntimeWarning** – game continue

start()
start processor

state = None

stop() → None
immediate stop game loop

progames.core.engine.producers module

```
class progames.core.engine.producers.Producer (store:    progames.core.engine.abc.Store,
                                              generator:    collec-
                                              tions.abc.AsyncGenerator)

    Bases: object

    produce_async ()

    start ()
        listen to inputs
```

progames.core.engine.stores module

```
class progames.core.engine.stores.SimpleQueueStore
    Bases: progames.core.engine.abc.Store

    empty ()
        check if the store is empty

    pop (*args, **kwargs)
        get the first item

        Returns the first item

    store (entity, *args, **kwargs)
        store item

        Parameters entity – item to store
```

progames.core.engine.worlds module

```
class progames.core.engine.worlds.AbstractGameWorld
    Bases: progames.core.engine.abc.GameWorld

    accept (command:    progames.core.engine.messages.Command)    →
        List[progames.core.engine.messages.Event]

        Parameters command – command

    process (command:    progames.core.engine.messages.Command)    →
        List[progames.core.engine.messages.Event]
        put your logic here

        Parameters command – the command

        Returns list of events

        Raises

            • RuntimeError – game crashed

            • RuntimeWarning – continue

    rollback (command: progames.core.engine.messages.Command) → None

        Parameters command – the failed command which causes rollback

    update () → None
        Some game need update game world each frame
```

validate (*command: progames.core.engine.messages.Command*)
 validate command

Raises `ValidationError` – invalid command

Module contents

```
class progames.core.engine.Command
    Bases: progames.core.engine.abc.Message
    base command

    visit (world: progames.core.engine.abc.GameWorld) → Iterable[progames.core.engine.messages.Event]

class progames.core.engine.Event
    Bases: progames.core.engine.abc.Message
    base event

class progames.core.engine.GameEnded
    Bases: progames.core.engine.messages.Event
    default event raise when game is ended

class progames.core.engine.GameWorld
    Bases: object

    accept (command: progames.core.engine.messages.Command) → Iter-
        able[progames.core.engine.messages.Event]
        Parameters command – command

    rollback (command: progames.core.engine.messages.Command) → None
        Parameters command – the failed command which causes rollback

    update () → None
        Some game need update game world each frame

class progames.core.engine.AbstractGameWorld
    Bases: progames.core.engine.abc.GameWorld

    accept (command: progames.core.engine.messages.Command) →
        List[progames.core.engine.messages.Event]
        Parameters command – command

    process (command: progames.core.engine.messages.Command) →
        List[progames.core.engine.messages.Event]
        put your logic here
        Parameters command – the command
        Returns list of events
        Raises
            • RuntimeError – game crashed
            • RuntimeWarning – continue

    rollback (command: progames.core.engine.messages.Command) → None
        Parameters command – the failed command which causes rollback
```

update () → None

Some game need update game world each frame

validate (*command: progames.core.engine.messages.Command*)

validate command

Raises `ValidationError` – invalid command

class `progames.core.engine.Producer` (*store: progames.core.engine.abc.Store, generator: collections.abc.AsyncGenerator*)

Bases: `object`

produce_async ()

start ()

listen to inputs

class `progames.core.engine.Processor` (*game: progames.core.engine.GameWorld, store: progames.core.engine.Store, *args, **kwargs*)

Bases: `object`

Process game and produce events

Notes

- Player's input data will be processed and transform to commands.
- Each frame, game only consume 1 command from command store.
- Command is a visitor, travel around game world and produce events. These events will be collected and sent to listeners.
- Rollback is required to implement

command_store = None

command store

cps = 50

commands per seconds (max), 0 < cps

get_next_command () → `typing.Optional[progames.core.engine.Command]`

get command to process

:raise `RuntimeWarning`

handle (*event: progames.core.engine.Event*) → None

base handle event, don't forget call: `super().handle(event)`

is_over () → bool

check if game is over

over = None

flag for checking if game is over

process_command () → None

process commands one by one

Flows:

1. pop 1 command from command store
2. the command go through game world, make changes and produce events
3. notify events to listeners (register)

4. update game world

Raises

- **RuntimeError** – game must stop
- **RuntimeWarning** – game continue

start()
start processor

state = None

stop() → None
immediate stop game loop

class `progames.core.engine.SimpleQueueStore`
Bases: `progames.core.engine.abc.Store`

empty()
check if the store is empty

pop(*args, **kwargs)
get the first item

Returns the first item

store(entity, *args, **kwargs)
store item

Parameters **entity** – item to store

Submodules

`progames.core.agents` module

class `progames.core.agents.AbstractPlayer`
Bases: `abc.ABC`

Abstract class for players, give basic functions which are required by flow

get_input_stream() → `io.IOBase`

get_output_stream() → `io.IOBase`

start()

stop()

`progames.core.exceptions` module

exception `progames.core.exceptions.ValidationError`
Bases: `Exception`

progames.core.loaders module

Module contents

progames.modules package

Subpackages

progames.modules.plaintext_mq package

Submodules

progames.modules.plaintext_mq.abc module

```
class progames.modules.plaintext_mq.abc.Subscriber
    Bases: object

    receive (*args, **kwargs)
```

progames.modules.plaintext_mq.core module

```
class progames.modules.plaintext_mq.core.Broker
    Bases: object

    broadcast (message)

    static deliver (receiver, message)

    static get_instance ()

    static hash (obj)
        hash stuffs

    publish (channel, message)

    subscribe (channel: str, subscriber)

    unsubscribe (channel, subscriber)
```

```
progames.modules.plaintext_mq.core.broadcast (message)
```

```
progames.modules.plaintext_mq.core.publish (channel, message)
```

```
progames.modules.plaintext_mq.core.subscribe (channel, subscriber)
```

```
progames.modules.plaintext_mq.core.unsubscribe (channel, subscriber)
```

Module contents

Module contents

3.1.2 Submodules

3.1.3 `progames.cli` module

Command Line Interface

`progames.cli.main()`

3.1.4 `progames.configurations` module

3.1.5 Module contents

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

p

- `progames`, 15
- `progames.cli`, 15
- `progames.configurations`, 15
- `progames.core`, 14
 - `agents`, 13
 - `engine`, 11
 - `abc`, 7
 - `concurrent`, 8
 - `messages`, 8
 - `processors`, 8
 - `producers`, 10
 - `stores`, 10
 - `worlds`, 10
 - `exceptions`, 13
 - `loaders`, 14
- `modules`, 15
 - `plaintext_mq`, 14
 - `abc`, 14
 - `core`, 14

A

AbstractGameWorld (class in *progames.core.engine*), 11
 AbstractGameWorld (class in *progames.core.engine.worlds*), 10
 AbstractPlayer (class in *progames.core.agents*), 13
 accept () (*progames.core.engine.abc.GameWorld* method), 7
 accept () (*progames.core.engine.AbstractGameWorld* method), 11
 accept () (*progames.core.engine.GameWorld* method), 11
 accept () (*progames.core.engine.worlds.AbstractGameWorld* method), 10

B

broadcast () (in module *progames.modules.plaintext_mq.core*), 14
 broadcast () (*progames.modules.plaintext_mq.core.Broker* method), 14
 Broker (class in *progames.modules.plaintext_mq.core*), 14

C

Command (class in *progames.core.engine*), 11
 Command (class in *progames.core.engine.messages*), 8
 command_store (*progames.core.engine.Processor* attribute), 12
 command_store (*progames.core.engine.processors.Processor* attribute), 9
 cps (*progames.core.engine.Processor* attribute), 12
 cps (*progames.core.engine.processors.Processor* attribute), 9

D

deliver () (*progames.modules.plaintext_mq.core.Broker* static method), 14

E

empty () (*progames.core.engine.abc.Store* method), 7

empty () (*progames.core.engine.SimpleQueueStore* method), 13
 empty () (*progames.core.engine.stores.SimpleQueueStore* method), 10

Event (class in *progames.core.engine*), 11
 Event (class in *progames.core.engine.messages*), 8

G

GameEnded (class in *progames.core.engine*), 11
 GameEnded (class in *progames.core.engine.messages*), 8
 GameWorld (class in *progames.core.engine*), 11
 GameWorld (class in *progames.core.engine.abc*), 7
 get_input_stream () (*progames.core.agents.AbstractPlayer* method), 13
 get_instance () (*progames.modules.plaintext_mq.core.Broker* static method), 14
 get_next_command () (*progames.core.engine.Processor* method), 12
 get_next_command () (*progames.core.engine.processors.Processor* method), 9
 get_output_stream () (*progames.core.agents.AbstractPlayer* method), 13

H

handle () (*progames.core.engine.Processor* method), 12
 handle () (*progames.core.engine.processors.Processor* method), 9
 hash () (*progames.modules.plaintext_mq.core.Broker* static method), 14

I

is_over () (*progames.core.engine.Processor* method), 12

`is_over()` (*progames.core.engine.processors.Processor* method), 9

M

`main()` (in module *progames.cli*), 15

`Message` (class in *progames.core.engine.abc*), 8

O

`over` (*progames.core.engine.Processor* attribute), 12

`over` (*progames.core.engine.processors.Processor* attribute), 9

P

`pop()` (*progames.core.engine.abc.Store* method), 7

`pop()` (*progames.core.engine.SimpleQueueStore* method), 13

`pop()` (*progames.core.engine.stores.SimpleQueueStore* method), 10

`process()` (*progames.core.engine.AbstractGameWorld* method), 11

`process()` (*progames.core.engine.worlds.AbstractGameWorld* method), 10

`process_command()` (*progames.core.engine.Processor* method), 12

`process_command()` (*progames.core.engine.processors.Processor* method), 9

`Processor` (class in *progames.core.engine*), 12

`Processor` (class in *progames.core.engine.processors*), 8

`produce_async()` (*progames.core.engine.Producer* method), 12

`produce_async()` (*progames.core.engine.producers.Producer* method), 10

`Producer` (class in *progames.core.engine*), 12

`Producer` (class in *progames.core.engine.producers*), 10

progames (module), 15

progames.cli (module), 15

progames.configurations (module), 15

progames.core (module), 14

progames.core.agents (module), 13

progames.core.engine (module), 11

progames.core.engine.abc (module), 7

progames.core.engine.concurrent (module), 8

progames.core.engine.messages (module), 8

progames.core.engine.processors (module), 8

progames.core.engine.producers (module), 10

progames.core.engine.stores (module), 10

progames.core.engine.worlds (module), 10

progames.core.exceptions (module), 13

progames.core.loaders (module), 14

progames.modules (module), 15

progames.modules.plaintext_mq (module), 14

progames.modules.plaintext_mq.abc (module), 14

progames.modules.plaintext_mq.core (module), 14

`publish()` (in module *progames.modules.plaintext_mq.core*), 14

`publish()` (*progames.modules.plaintext_mq.core.Broker* method), 14

R

`receive()` (*progames.modules.plaintext_mq.abc.Subscriber* method), 14

`rollback()` (*progames.core.engine.abc.GameWorld* method), 8

`rollback()` (*progames.core.engine.AbstractGameWorld* method), 11

`rollback()` (*progames.core.engine.GameWorld* method), 11

`rollback()` (*progames.core.engine.worlds.AbstractGameWorld* method), 10

`run()` (in module *progames.core.engine.concurrent*), 8

S

`SimpleQueueStore` (class in *progames.core.engine*), 13

`SimpleQueueStore` (class in *progames.core.engine.stores*), 10

`start()` (*progames.core.agents.AbstractPlayer* method), 13

`start()` (*progames.core.engine.Processor* method), 13

`start()` (*progames.core.engine.processors.Processor* method), 9

`start()` (*progames.core.engine.Producer* method), 12

`start()` (*progames.core.engine.producers.Producer* method), 10

`state` (*progames.core.engine.Processor* attribute), 13

`state` (*progames.core.engine.processors.Processor* attribute), 9

`stop()` (*progames.core.agents.AbstractPlayer* method), 13

`stop()` (*progames.core.engine.Processor* method), 13

`stop()` (*progames.core.engine.processors.Processor* method), 9

`Store` (class in *progames.core.engine.abc*), 7

`store()` (*progames.core.engine.abc.Store* method), 7

`store()` (*progames.core.engine.SimpleQueueStore* method), 13

`store()` (*progames.core.engine.stores.SimpleQueueStore* method), 10

```

subscribe() (in module
    progames.modules.plaintext_mq.core), 14
subscribe() (progames.modules.plaintext_mq.core.Broker
    method), 14
Subscriber (class in
    progames.modules.plaintext_mq.abc), 14

```

U

```

unsubscribe() (in module
    progames.modules.plaintext_mq.core), 14
unsubscribe() (progames.modules.plaintext_mq.core.Broker
    method), 14
update() (progames.core.engine.abc.GameWorld
    method), 8
update() (progames.core.engine.AbstractGameWorld
    method), 11
update() (progames.core.engine.GameWorld method),
    11
update() (progames.core.engine.worlds.AbstractGameWorld
    method), 10

```

V

```

validate() (progames.core.engine.AbstractGameWorld
    method), 12
validate() (progames.core.engine.worlds.AbstractGameWorld
    method), 10
ValidationError, 13
visit() (progames.core.engine.Command method), 11
visit() (progames.core.engine.messages.Command
    method), 8

```