PROMISES FOR UNITY

BY FISHERMAN PRODUCTION

QUICK SUMMARY

This is a c# implementation of the promise pattern for unity with optional automatic pooling. Contains additionally utils for sequences, races and parallel promise runs.

VERSION

1.0.0

HOW DO I GET SET UP?

Just copy the files to your project assets folder or use the .unityPackage file from the asset store.

CONFIGURATION

There is no in editor configuration, but static fields that can be set via code.

DEFAULTRETURNTOPOOL

Defines which value is used as default if Promise.Create will be called without any further specifications about pooling.
Easy way to activate/deactivate pooling globally. Default setting is true.

EVENT UNHANDLEDREJECTION

Will be triggered if a promise gets rejected but no rejection handler is registered. Add a logger to this event if you want to catch not handles rejections.

HOW TO USE IT?

CREATING A PROMISE

```
// non typed promise
    Promise promise = Promise.Create();

// typed promise
    Promise<int> intPromise = Promise<int>.Create();
```

RESOLVE / REJECT A PROMISE

```
// resolved non typed promise
    promise.Resolve();

// resolve typed promise
    intPromise.Resolve(42);

// maybe you want to check the status before
    if (promise.State == PromiseState.Pending)
    {
        promise.Resolve();
    }
```

REACT ON PROMISE RESULT

```
// you can use a lambda function but this will cause garbage due to boxing.
// non typed
    promise.Then(() => Debug.Log("Do something!"));

// typed
    intPromise.Then(value => Debug.Log("The result was " + value + ", do
something!"));

// better assign a method
// non typed
    promise.Then(OnPromiseResolved);

// typed
    intPromise.Then(OnIntPromiseResolved);

// same approach to listen for rejection to handles error cases
    promise.Then(() => Debug.Log("Everything is fine!"), exception =>
Debug.Log("Something went wrong: " + exception.Message));
```

```
    intPromise.Then(OnIntPromiseResolved, OnIntPromiseRejected);

// handling methods
    private void OnIntPromiseResolved(int value)
    {
        Debug.Log("The result was " + value + ", do something!");
    }
    private void OnPromiseResolved()
    {
        Debug.Log("Do something!");
    }
    private void OnIntPromiseRejected(Exception exception)
    {
        Debug.Log("Something went wrong: " + exception.Message);
    }
```

CLEANUP PROMISE

There are several ways to clean up promises. The recommend one would be to use pooling. In some specific cases where a typed promise is used rarely or only once a non-pooled promise would be the best choice to free up the memory.

RETURN TO POOL

```
// this will return the promise to the pooling ignoring the setting on
creation time

    promise.ReturnToPool();
    intPromise.ReturnToPool();
```

DESTROY

```
// this will destroy the promise by ignoring the setting on creation time.
Promise instance will be collected by GC.

    promise.Destroy();
    intPromise.Destroy();
```

LOOSE REFERENCE (NOT RECOMMEND, JUST FOR SAFETY)

```
// losing the reference is handled in this implementation and takes into
account the pooling setting on creation time. If pooling is active the
promise object will be caught before the garbage collector starts and
returns the promise into the pool. If not, the promise will be garbage
collected.
```

```
// If you do this with a pending promise, the promise will be rejected
first and all rejection listener will be called in both cases.

    promise = null;
    intPromise = null;
```

SEQUENCES

A sequence will call all listed method after each other. Only if the first method was successful resolved then the second will be called and so on.

```
// only parameter less methods can be in a sequence. If a certain parameter
set is needed for all methods another generic implementation of Sequence is
needed.
// non typed
    IPromise sequence = Promise.Sequence(true, MethodUsingPromise,
MethodUsingPromise, MethodUsingPromise);


    private IPromise MethodUsingPromise()
    {
        Promise promise = Promise.Create();
        // do something async and call promise.Resolve() after completion.
        return promise;
    }

// typed
    Promise<int>.Sequence(true, MethodUsingTypedResultPromise,
MethodUsingTypedResultPromise, MethodUsingTypedResultPromise);


    private IPromise<int> MethodUsingTypedResultPromise()
    {
        Promise<int> promise = Promise<int>.Create();
        // do something async and call promise.Resolve(value) after
completion.
        return promise;
    }
```

RACES

In a race the returned promise will be resolved as soon as one of the competed promises will be resolved.

```
    IPromise promise1 = Promise.Create();
    IPromise promise2 = Promise.Create();
```

```
    IPromise promise3 = Promise.Create();


    IPromise race = Promise.Race(true, promise1, promise2, promise3);
    race.Then(() => Debug.Log("One of them are done!"), exception =>
Debug.Log("One had a rejection!"));

// also possible using method directly like for sequences
    IPromise race2 = Promise.Race(true, MethodUsingPromise,
MethodUsingPromise, MethodUsingPromise);

// same for typed promises
    IPromise<int> race3 = Promise<int>.Race(true,
MethodUsingTypedResultPromise, MethodUsingTypedResultPromise,
MethodUsingTypedResultPromise);
```

ALL (PARALLEL)

The parallel promise behaves like a join or a funnel. Only if all promises are resolved the All promise will be resolved. You can use that to wait for all running processes.

```
    IPromise promise1 = Promise.Create();
    IPromise promise2 = Promise.Create();
    IPromise promise3 = Promise.Create();


    IPromise all = Promise.All(true, promise1, promise2, promise3);
    all.Then(() => Debug.Log("All of them are done now!"), exception =>
Debug.Log("One had a rejection!"));

// also possible using method directly like for sequences
    IPromise all2 = Promise.All(true, MethodUsingPromise,
MethodUsingPromise, MethodUsingPromise);

// special case for typed promises, you get back all results in a list
    IPromise<List<int>> all3 = Promise<int>.All(true,
MethodUsingTypedResultPromise, MethodUsingTypedResultPromise,
MethodUsingTypedResultPromise);
```

WHO DO I TALK TO?

Contact via Mail: unity@fisherman-production.de