

---

# **c-simulations Documentation**

*Release 0.3.3*

**Gregory McWhirter**

April 18, 2015



<b>1</b>	<b>Replicator Games</b>	<b>3</b>
1.1	Constants . . . . .	3
1.2	Types . . . . .	3
1.3	Functions . . . . .	5
<b>2</b>	<b>Replicator Populations</b>	<b>7</b>
2.1	Types . . . . .	7
2.2	Functions . . . . .	7
<b>3</b>	<b>Replicator Simulations</b>	<b>9</b>
3.1	Types . . . . .	9
3.2	Functions . . . . .	9
<b>4</b>	<b>Urn-Learning Games</b>	<b>11</b>
4.1	Types . . . . .	11
4.2	Functions . . . . .	11
<b>5</b>	<b>Urn-Learning Urns</b>	<b>13</b>
5.1	Types . . . . .	13
5.2	Functions . . . . .	13
<b>6</b>	<b>Urn-Learning Simulations</b>	<b>15</b>
6.1	Types . . . . .	15
6.2	Functions . . . . .	15
<b>7</b>	<b>Indices and tables</b>	<b>17</b>



This is a framework for running two types of game theory simulations in C. It can run n-population replicator dynamics simulations or Herrnstein-Roth-Erev urn learning simulations.

It builds both static and shared versions of two libraries that should be linked to the program that actually runs the simulations.

Contents:



---

## Replicator Games

---

The `replicator_game.c` and `replicator_game.h` files handle the functionality of defining the game being played for replicator dynamics simulations.

### 1.1 Constants

**CACHE\_NONE**

This constant indicates that nothing should be cached. It is a `cache_mask`.

**CACHE\_PROFILES**

This constant indicates that only the strategy profiles should be cached. It is a `cache_mask`.

**CACHE\_PAYOFFS**

This constant indicates that only the payoffs should be cached. It is a `cache_mask`.

**CACHE\_ALL**

This constant indicates that both the profiles and payoffs should be cached. It is equivalent to `CACHE_PROFILES | CACHE_PAYOFFS`. It is a `cache_mask`.

### 1.2 Types

**cache\_mask**

This type is an unsigned int. It is specially named to indicate that one of `CACHE_NONE`, `CACHE_PROFILES`, `CACHE_PAYOFFS`, or `CACHE_ALL` should be used.

`double (*payoff_function) (int players, int *strategy_profile)`

This type defines the signature for a payoff function that takes a strategy profile and returns an array of payoff values.

**strategyprofiles\_t**

This is an alias for the `StrategyProfiles` struct.

**game\_t**

This is an alias for the `Game` struct.

**payoffcache\_t**

This is an alias for the `PayoffCache` struct.

struct **StrategyProfiles**

This struct holds the information for strategy profiles (tuples of interaction possibility)

int **StrategyProfiles.count**

This is the number of profiles the struct holds.

int **StrategyProfiles.size**

This is the size of each profile.

int\* **StrategyProfiles.types**

This is an array of types (the number of types for player *i* is in the *i*th entry). The `size` member indicates the size of this array.

int **StrategyProfiles.has\_cached\_info**

This is a flag to indicate whether the struct has cached information stored. It is used for the `StrategyProfiles_destroy()` function.

int\*\* **StrategyProfiles.profiles**

This is an array of the possible strategy profiles. It has `size` stored in `count` and each element has `size` from the `size` member.

int\*\*\* **StrategyProfiles.player\_strategy\_profiles**

This is an array of the possible strategy profiles sorted by players participating in them. It has the size defined by the `size` member. The first dimension corresponds to the player. The second dimension corresponds to the number of strategies for that player (`size count/types[i]`). The third dimension is a list of the profiles in which that strategy participates, represented by indices referring to the `StrategyProfiles.profiles` list.

struct **Game**

This struct holds data about the game being played under the dynamics.

int **Game.populations**

This is how many populations the game has.

int **Game.players**

This is how many players there are in the game.

int\* **Game.types**

This is a list, for each player, how many strategies that player has.

`payoff_function` **Game.payoffs**

This is the function that returns a payoff vector for a certain strategy profile in the game.

struct **PayoffCache**

This is a struct that holds a cache of pre-calculated payoff vectors.

int **PayoffCache.count**

This is how many items are in the cache.

int **PayoffCache.has\_cached\_info**

This is a flag to indicate that the cache has information in it that should be freed.

int **PayoffCache.free\_profiles**

This is a flag to indicate that the profiles are cached and should be freed.

`payoff_function` **PayoffCache.payoffs**

This is the payoff function that generates the payoffs.

`strategyprofiles_t*` **PayoffCache.profiles**

This is the pointer to the cache of strategy profiles.

double\*\* **PayoffCache.payoff\_cache**

This is the cache of payoff vectors. Each payoff vector is an array of doubles, and the collection is an array of those arrays.

## 1.3 Functions

### 1.3.1 StrategyProfiles

`strategyprofiles_t * StrategyProfiles_create (int players, int *types, cache_mask cache)`

This creates a `strategyprofiles_t` struct for the requisite number of players.

The `types` parameter is a list of number of strategies that each player has.

The `cache` parameter is how much of the profiles and payoffs to cache.

`int * StrategyProfiles_getProfile (strategyprofiles_t *sprofs, int num)`

This returns the strategy profile corresponding to the `num`'th entry in the `:c:data: `sprofs` array.

`int * StrategyProfiles_getPlayerProfile (strategyprofiles_t *sprofs, int player, int strategy, int num)`

This returns the `num`'th strategy profile that player `:c:data: `player`'s strategy strategy is involved in.

`int StrategyProfiles_getPlayerProfileNumber (strategyprofiles_t *sprofs, int player, int strategy, int num)`

This returns the index in the profile list of `sprofs` that the `num`'th strategy profile of player `:c:data: `player`'s strategy strategy is involved in.

`void StrategyProfiles_destroy (strategyprofiles_t *sprofs)`

This frees all data associated with `sprofs`.

### 1.3.2 Game

`game_t * Game_create (int players, int populations, int *types, payoff_function payoffs)`

This creates a `game_t` struct based on the requested data.

The number of populations must either be 1 or equal to the number of players.

The parameter `types` is a list of the number of strategies for each player.

The parameter `payoffs` is the payoff function for the game.

`void Game_destroy (game_t *game)`

This frees all data associated with `game`.

`strategyprofiles_t * Game_StrategyProfiles_create (game_t *game, cache_mask cache)`

This creates a `strategyprofiles_t` struct from the data already present in a `game_t` struct.

### 1.3.3 PayoffCache

`payoffcache_t * PayoffCache_create (game_t *game, strategyprofiles_t *profiles, cache_mask do_cache)`

This creates a `payoffcache_t` struct based on the provided information.

`double * PayoffCache_getPayoffs (payoffcache_t *cache, int profile_index)`

This returns the payoffs for the cached profile index `profile_index`.

`void PayoffCache_destroy (payoffcache_t *cache)`

This frees all data associated with `cache`.



---

## Replicator Populations

---

### 2.1 Types

**population\_t**

This is a shortcut for a `Population` struct.

**popcollection\_t**

This is a shortcut for a `PopCollection` struct.

struct **Population**

This struct holds the data relevant to a single replicator population

int **Population.size**

This member determines how many entries are in the `Population.proportions` array.

double\* **Population.proportions**

This member holds the population proportions.

struct **PopCollection**

This struct holds a collection of `population_t` structs.

int **PopCollection.size**

This determines how many populations are collected.

int\* **PopCollection.pop\_sizes**

This is an array of the sizes of each of the populations collected.

`popcollection_t`\*\* **PopCollection.populations**

This is an array of the populations.

### 2.2 Functions

#### 2.2.1 Population

```
population_t * Population_create (int size)
```

```
void Population_destroy (population_t *pop)
```

```
int Population_equal (population_t *pop1, population_t *pop2, double effective_zero)
```

```
void Population_copy (population_t *target, population_t *source)
```

```
void Population_randomize (population_t *pop)
```

```
void Population_serialize (population_t *pop, FILE * target_file)
population_t * Population_deserialize (FILE * source_file)
```

## 2.2.2 PopCollection

```
popcollection_t * PopCollection_create (int num_pops, int *sizes)
popcollection_t * PopCollection_clone (popcollection_t *original)
void PopCollection_destroy (popcollection_t *coll)
int PopCollection_equal (popcollection_t *coll1, popcollection_t *coll2, double effective_zero)
void PopCollection_copy (popcollection_t *target, popcollection_t *source)
void PopCollection_randomize (popcollection_t *coll)
void PopCollection_serialize (popcollection_t *coll, FILE * target_file)
popcollection_t * PopCollection_deserialize (FILE * source_file)
```

---

## Replicator Simulations

---

### 3.1 Types

```
void (*cb_func) (game_t *game, int generation, popcollection_t *generation_pop, FILE *out-  
file)
```

### 3.2 Functions

```
void replicator_dynamics_setup ()
```

```
popcollection_t * replicator_dynamics (game_t *game, popcollection_t *start_pops,  
double alpha, double effective_zero,  
int max_generations, cache_mask caching,  
cb_func on_generation, FILE *outfile)
```

```
double earned_payoff (int player, int strategy, popcollection_t *pops, strategyprofiles_t *pro-  
files, payoffcache_t *payoff_cache)
```

```
double average_earned_payoff (int player, popcollection_t *pops, strategyprofiles_t *pro-  
files, payoffcache_t *payoff_cache)
```

```
void update_population_proportions (double alpha, int player, population_t *pop, pop-  
collection_t *curr_pops, strategyprofiles_t *pro-  
files, payoffcache_t *payoff_cache, int *threads)
```



---

## Urn-Learning Games

---

### 4.1 Types

```
unsigned int * (urn_interaction) (unsigned int players, urncollection_t **player_urns,  
                                   rk_state *random_state)
```

```
urngame_t
```

```
struct UrnGame
```

```
    unsigned int UrnGame.num_players
```

```
    unsigned int** UrnGame.types
```

```
    urncollection_t** UrnGame.player_urns
```

```
    urn_interaction UrnGame.interaction_function
```

### 4.2 Functions

```
urngame_t* UrnGame_create (unsigned int players, unsigned int *num_urns, unsigned  
                           int **types, double ***initial_counts, urn_interaction func)
```

```
void UrnGame_destroy (urngame_t *urngame)
```

```
unsigned int * default_urnlearning_interaction (unsigned int players, urn-  
                                                collection_t **player_urns,  
                                                rk_state *rand_state_ptr)
```

```
void UrnGame_copy (urngame_t *source, urngame_t *target)
```

```
urngame_t * UrnGame_clone (urngame_t *urngame)
```



---

## Urn-Learning Urns

---

### 5.1 Types

```
urn_t
urncollection_t
struct Urn
    unsigned int Urn.types
    double* Urn.counts
    double* Urn.proportions
struct UrnCollection
    unsigned int UrnCollection.num_urns
    urn_t** UrnCollection.urns
```

### 5.2 Functions

#### 5.2.1 Urn

```
urn_t* Urn_create (unsigned int types, double *initial_counts)
void Urn_destroy (urn_t *urn)
void Urn_update (urn_t *urn, double *count_updates)
unsigned int Urn_select (urn_t *urn, double random_draw)
unsigned int Urn_randomSelect (urn_t *urn, rk_state *rand_state_ptr)
void Urn_display (urn_t *urn, char *prefix, FILE *outfile)
urn_t* Urn_clone (urn_t *urn)
void Urn_copy (urn_t *source, urn_t *target)
```

## 5.2.2 UrnCollection

```
urncollection_t * UrnCollection_create (unsigned int num_urns, unsigned int * types, double **initial_counts)  
void UrnCollection_destroy (urncollection_t *urnc)  
urncollection_t * UrnCollection_clone (urncollection_t *urnc)  
void UrnCollection_copy (urncollection_t *source, urncollection_t *target)
```

---

## Urn-Learning Simulations

---

### 6.1 Types

```
double ** (*payoff_function) (unsigned int players, unsigned int **types, unsigned int  
                               * state_action_profile)
```

### 6.2 Functions

```
void urnlearning_dynamics (urngame_t *urngame, unsigned long max_iterations, pay-  
                           off_function payoffs)
```

Examples:

Universal Deception simulations (Replicator)

Self-Deception simulations (Urn-Learning)



---

## Indices and tables

---

- *genindex*
- *modindex*
- *search*



**A**

average\_earned\_payoff (C function), 9

**C**

CACHE\_ALL (C macro), 3  
 cache\_mask (C type), 3  
 CACHE\_NONE (C macro), 3  
 CACHE\_PAYOFFS (C macro), 3  
 CACHE\_PROFILES (C macro), 3  
 cb\_func (C type), 9

**D**

default\_urnlearning\_interaction (C function), 11

**E**

earned\_payoff (C function), 9

**G**

Game (C type), 4  
 Game.Game.payoffs (C member), 4  
 Game.Game.players (C member), 4  
 Game.Game.populations (C member), 4  
 Game.Game.types (C member), 4  
 Game\_create (C function), 5  
 Game\_destroy (C function), 5  
 Game\_StrategyProfiles\_create (C function), 5  
 game\_t (C type), 3

**P**

payoff\_function (C type), 3, 15  
 PayoffCache (C type), 4  
 PayoffCache.PayoffCache.count (C member), 4  
 PayoffCache.PayoffCache.free\_profiles (C member), 4  
 PayoffCache.PayoffCache.has\_cached\_info (C member), 4  
 PayoffCache.PayoffCache.payoff\_cache (C member), 4  
 PayoffCache.PayoffCache.payoffs (C member), 4  
 PayoffCache.PayoffCache.profiles (C member), 4  
 PayoffCache\_create (C function), 5  
 PayoffCache\_destroy (C function), 5

PayoffCache\_getPayoffs (C function), 5  
 payoffcache\_t (C type), 3  
 PopCollection (C type), 7  
 PopCollection.PopCollection.pop\_sizes (C member), 7  
 PopCollection.PopCollection.populations (C member), 7  
 PopCollection.PopCollection.size (C member), 7  
 PopCollection\_clone (C function), 8  
 PopCollection\_copy (C function), 8  
 PopCollection\_create (C function), 8  
 PopCollection\_deserialize (C function), 8  
 PopCollection\_destroy (C function), 8  
 PopCollection\_equal (C function), 8  
 PopCollection\_randomize (C function), 8  
 PopCollection\_serialize (C function), 8  
 popcollection\_t (C type), 7  
 Population (C type), 7  
 Population.Population.proportions (C member), 7  
 Population.Population.size (C member), 7  
 Population\_copy (C function), 7  
 Population\_create (C function), 7  
 Population\_deserialize (C function), 8  
 Population\_destroy (C function), 7  
 Population\_equal (C function), 7  
 Population\_randomize (C function), 7  
 Population\_serialize (C function), 7  
 population\_t (C type), 7

**R**

replicator\_dynamics (C function), 9  
 replicator\_dynamics\_setup (C function), 9

**S**

StrategyProfiles (C type), 3  
 StrategyProfiles.StrategyProfiles.count (C member), 3  
 StrategyProfiles.StrategyProfiles.has\_cached\_info (C member), 4  
 StrategyProfiles.StrategyProfiles.player\_strategy\_profiles (C member), 4  
 StrategyProfiles.StrategyProfiles.profiles (C member), 4  
 StrategyProfiles.StrategyProfiles.size (C member), 4

StrategyProfiles.StrategyProfiles.types (C member), 4  
StrategyProfiles\_create (C function), 5  
StrategyProfiles\_destroy (C function), 5  
StrategyProfiles\_getPlayerProfile (C function), 5  
StrategyProfiles\_getPlayerProfileNumber (C function), 5  
StrategyProfiles\_getProfile (C function), 5  
strategyprofiles\_t (C type), 3

## U

update\_population\_proportions (C function), 9  
Urn (C type), 13  
Urn.Urn.counts (C member), 13  
Urn.Urn.proportions (C member), 13  
Urn.Urn.types (C member), 13  
Urn\_clone (C function), 13  
Urn\_copy (C function), 13  
Urn\_create (C function), 13  
Urn\_destroy (C function), 13  
Urn\_display (C function), 13  
urn\_interaction (C type), 11  
Urn\_randomSelect (C function), 13  
Urn\_select (C function), 13  
urn\_t (C type), 13  
Urn\_update (C function), 13  
UrnCollection (C type), 13  
UrnCollection.UrnCollection.num\_urns (C member), 13  
UrnCollection.UrnCollection.urns (C member), 13  
UrnCollection\_clone (C function), 14  
UrnCollection\_copy (C function), 14  
UrnCollection\_create (C function), 14  
UrnCollection\_destroy (C function), 14  
urncollection\_t (C type), 13  
UrnGame (C type), 11  
UrnGame.UrnGame.interaction\_function (C member), 11  
UrnGame.UrnGame.num\_players (C member), 11  
UrnGame.UrnGame.player\_urns (C member), 11  
UrnGame.UrnGame.types (C member), 11  
UrnGame\_clone (C function), 11  
UrnGame\_copy (C function), 11  
UrnGame\_create (C function), 11  
UrnGame\_destroy (C function), 11  
urngame\_t (C type), 11  
urnlearning\_dynamics (C function), 15